

User's guide to WPP version 2.2

N. Anders Petersson¹

Björn Sjögreen¹

September 17, 2013

¹Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, PO Box 808, Livermore CA 94551. This is contribution LLNL-SM-487431.

Disclaimer This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Auspices Statement This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

Contents

1	Introduction	5
2	Getting started	7
2.1	Running <i>WPP</i>	7
3	Coordinate system, units and the grid	9
3.1	Geographical coordinates	10
4	Sources, time-functions and grid sizes	12
4.1	Sources and time-functions in <i>WPP</i>	12
4.2	Predefined time functions	13
4.2.1	Gaussian	14
4.2.2	GaussianInt	14
4.2.3	Ricker	15
4.2.4	RickerInt	15
4.2.5	Brune	15
4.2.6	BruneSmoothed	16
4.2.7	Liu	17
4.2.8	Triangle	18
4.2.9	Sawtooth	18
4.2.10	Ramp	18
4.2.11	Smoothwave	19
4.2.12	VerySmoothBump	19
4.2.13	GuassianWindow	19
4.3	How fine does the grid need to be?	21
4.3.1	Lamb's problem	22
5	The material model	24
5.1	The block command	25
5.2	The efile command	26
5.3	The pfile command	27
5.4	The ifile command	30
6	Topography	31
6.1	Gaussian hill topography	32
6.2	Topography grid file	32

6.3	Etree topography	33
7	Mesh refinement	35
8	Attenuation	38
8.1	Viscoelastic modeling	38
9	Output options	41
9.1	Setting the output directory	41
9.2	Time-history at a receiver station: the sac command	41
9.3	2-D cross-sectional data: the image command	44
9.4	Generating a bird's eye view of the problem domain: the gmt command	45
10	Examples	47
10.1	Lamb's problem	47
10.2	Examples from Lifelines project 1A01: Validation of basin response codes	48
10.2.1	The LOH.1 problem	48
10.2.2	The LOH.2 problem	51
10.2.3	The LOH.3 problem	51
10.3	The Grenoble basin test case	53
10.4	Modeling the October 2007, Alum Rock earthquake	56
10.5	A scenario earthquake in the Caucasus region	59
11	Keywords in the input file	64
11.1	Basic commands	64
11.1.1	fileio [optional]	64
11.1.2	grid [required]	65
11.1.3	time [required]	66
11.1.4	source [required]	66
11.1.5	prefilter [optional]	68
11.2	The material model [required]	69
11.2.1	attenuation [optional]	69
11.2.2	block	70
11.2.3	efile	71
11.2.4	pfile	72
11.2.5	ifile	73
11.2.6	material	73
11.2.7	globalmaterial [optional]	74
11.3	Topography and mesh refinement [optional]	74
11.3.1	topography [optional]	74
11.3.2	refinement [optional]	75
11.4	Output commands [optional]	76
11.4.1	sac [optional]	76
11.4.2	image [optional]	77
11.4.3	volimage [optional]	79
11.4.4	gmt [optional]	82
11.5	WPP testing commands [optional]	82

11.5.1	twilight	82
11.5.2	testlamb	82
11.5.3	testpointsource	83
11.6	Advanced simulation controls [optional]	84
11.6.1	supergrid [optional]	84
11.6.2	boundary_conditions [optional]	84
11.6.3	developer [optional]	85
12	File formats	87
12.1	topography	87
12.1.1	topography on a geographic lattice	87
12.1.2	topography on a Cartesian lattice	88
12.2	pfile	89
12.2.1	pfile on a geographic lattice	89
12.2.2	pfile on a Cartesian lattice	90
12.3	ifile	91
12.4	sac	92
12.5	image	93
12.6	volimage	94
13	Run time and memory requirements	96
13.1	Run time	96
13.2	Memory usage	96
A	Installing <i>WPP</i>	99
A.1	Supported platforms and compilers	99
A.2	MPI and other third party libraries	99
A.2.1	Mac computers	100
A.2.2	Linux machines	100
A.3	Directory structure	101
A.4	Compiling and Linking <i>WPP</i> (without the cencalvm library)	102
A.4.1	How do I setup the make.inc file?	103
A.5	Installing cencalvm and its supporting libraries	103
B	Testing the <i>WPP</i> installation	106
B.1	Method of manufactured solutions	106
B.2	Lamb's problem	108

Chapter 1

Introduction

WPP is a computer program for simulating seismic wave propagation on parallel machines. *WPP* solves the governing equations in second order formulation using a node-based finite difference approach [12]. The numerical method uses summation by parts stencils to guarantee stability of the time-stepping procedure. The stability holds for heterogeneous material models on Cartesian and curvilinear grids, free surface boundary conditions, mesh refinement interfaces with hanging nodes, and for heterogeneous visco-elastic materials.

WPP implements substantial capabilities for 3-D seismic modeling, with a free surface condition on the top boundary, non-reflecting far-field boundary conditions on the other boundaries, point force and point moment tensor source terms with many predefined time dependencies, fully 3-D heterogeneous material model specification, elastic or visco-elastic materials, output of synthetic seismograms in the *SAC* [6] format, output of *GMT* [15] scripts for laying out simulation information on a map, and output of (derived quantities of) the solution field as well as the material model, both as 2-D slices and 3-D volumetric data.

Version 2.2 of *WPP* allows the free surface boundary condition to be imposed on a curved topography. For this purpose a curvilinear mesh is used near the free surface, extending into the computational domain to a user specified level. The (visco-) elastic wave equation and the free surface boundary conditions are discretized on the curvilinear mesh using the energy conserving technique described in [2]. A curvilinear mesh generator is built into *WPP* and the curvilinear mesh is automatically generated from the description of the topography. Below the curvilinear grid, the (visco-) elastic wave equation is discretized on Cartesian meshes, which leads to a more computationally efficient algorithm.

In version 2.2 of *WPP*, Cartesian local mesh refinement can be used to make the computational mesh finer near the free surface, where more resolution often is needed to resolve short wave lengths in the solution, for example in sedimentary basins. The mesh refinement is performed in the vertical direction and each Cartesian grid is constructed from user specified refinement levels. In this approach, the grid size in all three spatial directions is doubled across each mesh refinement interface, leading to substantial savings in memory and computational effort. The energy conserving mesh refinement coupling method described in [13] is used to handle the hanging nodes along the refinement interface.

Visco-elastic behavior can be important when modeling the dissipative nature of realistic materials, especially for higher frequencies. Version 2.2 of *WPP* uses the rheological model of standard linear solid (SLS) elements, coupled in parallel. The coefficients in each SLS are determined such that the resulting quality factors Q_p and Q_s , for the attenuation of P- and S-waves, become ap-

proximately constant as function of frequency. These quality factors can vary from grid point to grid point over the computational domain and are read in the same way as the elastic properties of the material model. The underlying numerical method for solving the visco-elastic wave equation is described in [14].

The **examples** subdirectory of the *WPP* source distribution contains several examples and validation tests. Many Matlab/octave scripts are provided in the **tools** directory.

Acknowledgments

Many people have contributed to the development of *WPP* and we would like to thank (in no particular order) Artie Rodgers, Heinz-Otto Kreiss, Stefan Nilsson, Kathleen McCandless, Hrvoje Tkalčić, Steve Blair, Daniel Appelö, and Caroline Bono. This work was enabled by financial support from a Laboratory Directed Research and Development (LDRD) project at Lawrence Livermore National Laboratory, as well as support from the OASCR program at the Office of Science at the U.S. Department of Energy.

Chapter 2

Getting started

2.1 Running *WPP*

WPP can be run from the UNIX prompt or from a script. Normally *WPP* uses one argument, which is the name of the input file. The input file is an ASCII text file that contains a number of commands specifying the properties of the simulation, such as the dimensions of the computational domain, grid spacing, the duration of the simulation, the material properties, the source model, as well as the desired output. To improve readability of this document we have used the continuation character “\” to extend long commands to the subsequent line. There is however no support for continuation characters in *WPP*, so each command must be given on one (sometimes long) line in the input file.

Since *WPP* is a parallel code, it is required to be run under a parallel execution environment such as `mpiexec`, `mpirun`, or `srun`. With the exception of `srun`, which always is used on the Livermore computing parallel machines, it is important to start *WPP* with the correct parallel execution tool. Also note that some systems require you to start an `mpd` daemon before running any parallel programs (see `mpich2-doc-user.pdf` for more info). Refer to your local system administrator for information on how parallel programs should be started on your system.

Throughout this document we use the convention that input files have the file suffix `.in`. However, *WPP* does not discriminate based on file extension and will gladly attempt to read an input file regardless of its extension.

If your system is setup for using `mpiexec`, the command

```
shell> mpiexec -np 2 wpp test.in
```

runs *WPP* on 2 processes, and tells it to read input from a file named `test.in`. If you are using `mpirun`, you would instead use the command

```
shell> mpirun -np 2 wpp test.in
```

Remark: If *WPP* produces strange looking outputs where the same text is repeated several times (e.g. once per processor), you are probably running *WPP* under the wrong parallel execution environment.

Running on the Livermore Computing parallel linux clusters The `srun` command is currently used to run parallel jobs on LC machines. For example, the command


```
shell> srun -ppdebug -n 32 wpp xxx.in
```

runs *WPP* on 32 processors on the debug partition using *xxx.in* as the input file. Note that the *pdebug* partition is intended for shorter jobs and is subject to both a CPU time limit and a limit on the number of processors per job. Jobs requiring more computer resources must be submitted through the batch system, currently using the *msub* command. Refer to the Livermore Computing web pages for detailed information (<https://computing.llnl.gov>).

Version information (-v) Version information for the *WPP* executable can be obtained through *-v* flag:

```
[yorkville:~/src/wpp/optimize_v2.1] petersson1% wpp -v
```

```
-----  
                WPP Version 2.1  
Copyright (C) 2007-2011 Lawrence Livermore National Security, LLC.  
  
WPP comes with ABSOLUTELY NO WARRANTY; released under GPL.  
This is free software, and you are welcome to redistribute  
it under certain conditions, see LICENSE.txt for more details  
-----  
Compiled: Mon May 16 14:16:18 2011  
By:      petersson1  
Machine: yorkville.llnl.gov  
Compiler: /Users/petersson1/MacPorts/bin/mpicxx  
3rd party software base directory: /Users/petersson1/  
-----
```

Note that the same information is by default printed to standard out at the beginning of every run.

Chapter 3

Coordinate system, units and the grid

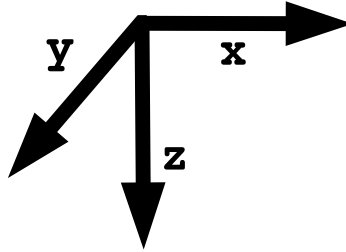


Figure 3.1: *WPP* uses a right handed coordinate system with the z-axis pointing downwards.

WPP uses a right-handed Cartesian coordinate system with the z-direction pointing downwards into the medium, see figure 3.1. *WPP* employs MKS (meters-kilograms-seconds) units; all distances (e.g., grid dimensions, spacing, and displacements) are in meters (m), time is in seconds (s), seismic P- and S-wave velocities are in meters per second (m/s), densities are in kilogram per cubic meter (kg/m^3), forces are in Newton (N), and seismic moment (torque) is in Newton-meters (Nm). All angles (e.g. latitude, longitude, azimuth, strike, dip and rake) are in degrees. The quality factors Q_P and Q_S are dimensionless.

In *WPP* the computational domain is rectangular in the horizontal plane,

$$0 \leq x \leq x_{max}, \quad 0 \leq y \leq y_{max}.$$

The topography surface

$$z = \tau(x, y),$$

defines the shape of the top surface in the vertical direction. *WPP* can also be run without topography, in which case $\tau(x, y) = 0$. The computational domain is given by

$$0 \leq x \leq x_{max}, \quad 0 \leq y \leq y_{max}, \quad \tau(x, y) \leq z \leq z_{max}. \quad (3.1)$$

The grid command in the input file specifies the extent of the computational domain and the grid size h . When mesh refinement is enabled, this is the grid size in the coarsest grid. The most obvious way of specifying the grid is by providing the number of grid points in each direction as well as the grid size,

```
grid nx=301 ny=201 nz=101 h=500.0
```

This line gives a grid with grid size 500 meters, which extends 150 km in x , 100 km in y and 50 km in the z -direction. Alternatively, the grid can be specified by giving the spatial range in each of the three dimensions and explicitly specifying the grid spacing. For example,

```
grid x=30e3 y=20e3 z=10e3 h=500.0
```

results in a grid which spans 30,000 meters in x , 20,000 meters in y , and 10,000 meters in the z -direction. The grid spacing is 500 meters, which is used to compute the number of grid points in each direction: $nx=61$, $ny=41$, and $nz=21$, for a total of 52,521 grid points. Note that the number of grid points in the different directions will be rounded to the nearest integer value according to the pseudo C-code

$$nx = (\text{int})(1.5 + x/h). \quad (3.2)$$

The extent in the x -direction is thereafter adjusted to

$$x = (nx - 1)h. \quad (3.3)$$

A corresponding procedure is performed in the other coordinate directions.

The third option is to give the spatial range in each of the three dimensions and specify the number of grid points in a particular direction:

```
grid x=30000 y=20000 z=10000 nx=100
```

In this case, the grid spacing is computed as

$$h = x/(nx - 1) = 303.03.$$

Note that no rounding needs to take place in this case, since h is a floating point number. Given this value of h , ny and nz are computed using formulas corresponding to (3.2) giving $ny=34$ and $nz=67$, for a total of 227,800 grid points. Again, the extents in the y and z -directions are adjusted corresponding to (3.3). The syntax for the grid command is given in Section 11.1.2.

3.1 Geographical coordinates

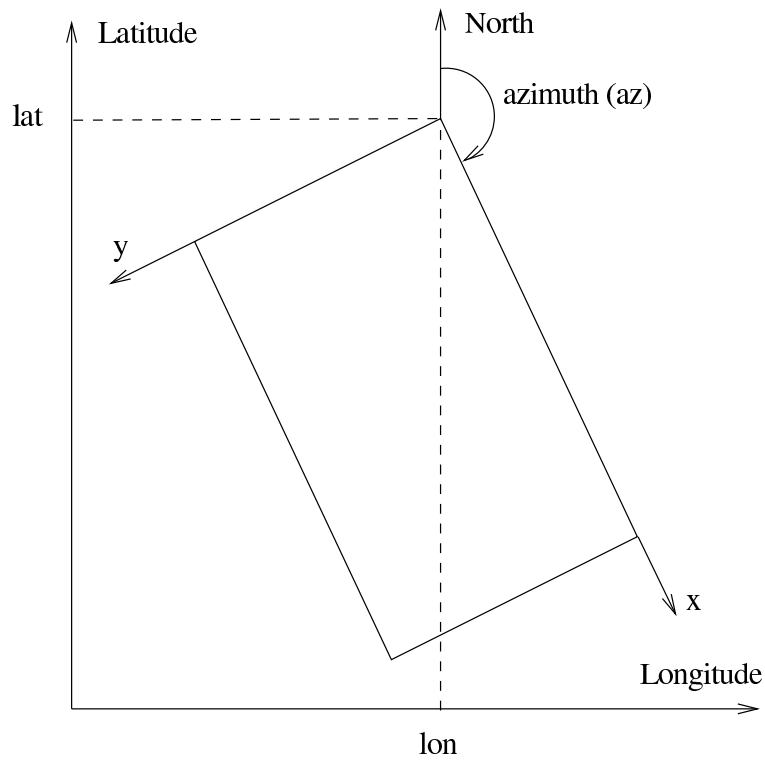
WPP supports geographical coordinates as an alternative way of specifying spatial locations, see Figure 3.2. The location of the origin of the Cartesian coordinate system is specified in the grid command, and if no location is given it defaults to latitude 37 degrees (North), longitude -118 degrees (West), with a 135 degree azimuthal angle from North to the x -axis. The vertical coordinate is zero ($z = 0$) at mean sea level. The latitude (ϕ) and longitude (θ) are calculated using the approximative formulae (where lat , lon , az , ϕ , and θ are in degrees)

$$\phi = \text{lat} + \frac{x \cos(\alpha) - y \sin(\alpha)}{M}, \quad \alpha = \text{az} \frac{\pi}{180}, \quad (3.4)$$

$$\theta = \text{lon} + \frac{x \sin(\alpha) + y \cos(\alpha)}{M \cos(\phi\pi/180)}, \quad (3.5)$$

and $M = 111,319.5$ meters/degree¹. You can change the location and orientation of the grid by specifying the latitude and longitude of the grid origin, and the azimuthal angle between North and the x -axis. For example:

¹Note that $M/60 = 1,855.325$ meters/minute, which corresponds to one minute of arc of longitude along the Equator on the WGS 84 ellipsoid. This distance is also known as a geographical mile and is approximately equal to a Nautical mile (1,852 meters).

Figure 3.2: Geographical coordinates in *WPP*.

```
grid h=500.0 x=30000.0 y=20000.0 z=10000.0 lat=39.0 lon=-117.0 az=150
```

sets the origin of the grid to latitude 39 degrees (North), longitude -117 degrees (West), and azimuthal angle 150 degrees.

Chapter 4

Sources, time-functions and grid sizes

4.1 Sources and time-functions in *WPP*

WPP solves the elastic (or visco-elastic) wave equation in second order formulation,

$$\begin{aligned}\rho \mathbf{u}_{tt} &= \nabla \cdot \mathcal{T} + \mathbf{F}(\mathbf{x}, t), & \mathbf{x} \text{ in } \Omega, \ t \geq 0, \\ \mathbf{u}(\mathbf{x}, 0) &= 0, \quad \mathbf{u}_t(\mathbf{x}, 0) = 0, & \mathbf{x} \text{ in } \Omega,\end{aligned}$$

where ρ is the density, $\mathbf{u}(\mathbf{x}, t)$ is the displacement vector, and \mathcal{T} is the stress tensor. The computational domain Ω is the box shaped region of equation (3.1). By default, a free surface (zero traction) boundary condition is enforced along the top boundary,

$$\mathcal{T} \cdot \mathbf{n} = 0, \quad z = \tau(x, y), \ t \geq 0,$$

where \mathbf{n} is the unit normal of the $z = \tau(x, y)$ surface. A super-grid damping layer surrounds the computational domain on all other sides of the computational domain.

The forcing term \mathbf{F} consists of a sum of point forces and point moment tensor source terms. For a point forcing we have

$$\mathbf{F}(\mathbf{x}, t) = g(t, t_0, \omega) F_0 \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} \delta(\mathbf{x} - \mathbf{x}_0),$$

where $\mathbf{x}_0 = (x_0, y_0, z_0)$ is the location of the point force in space, and $g(t, t_0, \omega)$ is the time function, with offset time t_0 and frequency parameter ω . The $(F_x, F_y, F_z)^T$ vector holds the Cartesian components of the force vector, which is scaled by the force amplitude F_0 .

For a moment tensor source we have

$$\mathbf{F}(\mathbf{x}, t) = g(t, t_0, \omega) M_0 \mathcal{M} \cdot \nabla \delta(\mathbf{x} - \mathbf{x}_0), \quad \mathcal{M} = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{pmatrix}.$$

In this case the seismic moment of the moment tensor is M_0 , otherwise the notation is the same as for a point force. Note that the moment tensor always is symmetric. A convenient way of specifying

moment sources is by using the dip, strike, and rake angles (see Section 11.1.4 for syntax) defined in Aki and Richards [1]. In this case, the total seismic moment $\sum M_0$ [Nm] is related to the moment magnitude by the formula

$$M_W = \frac{2}{3} \left[\log_{10} \left(\sum M_0 \right) - 9.1 \right].$$

After parsing all source commands in an input file, *WPP* outputs the moment magnitude using this formula. This information is given right before the time-stepping is started and looks like this:

```
-----
Total seismic moment (M0): 1.7162e+17 Nm
Moment magnitude      (Mw): 5.42305
Number of sources 542
-----
```

Note that the calculation of the seismic moment and magnitude assume that all sources are specified in terms of dip, strike, and rake angles. This guarantees that the matrix \mathcal{M} is properly normalized.

For moment tensor sources, the function $g(t)$ is called the moment history time function, while its time derivative $g'(t)$ is known as the moment rate time function. *WPP* calculates the displacements of the motion corresponding to the moment history time function $g(t)$. However, since the material properties are independent of time, the equations solved by *WPP* also govern the velocities when the time function is replaced by $g'(t)$, i.e., the corresponding moment rate time function. For example, if the solution calculated with the `GaussianInt` time function represents the displacements of the motion, the solution calculated with the `Gaussian` time function corresponds to the velocities of the same motion. Hence, if you are primarily interested in calculating velocities, you can reduce the amount of post processing by using the corresponding moment rate time function in the source term(s).

Note that most first order formulation codes (such as *E3D*) are based on the velocity-stress formulation of the elastic wave equation. These codes use the moment rate time function (i.e., the `Gaussian` time function in the above example) and solve for the velocities of the motion.

In *WPP* the forcing is specified in the input file using the `source` command. There needs to be at least one source command in the input file in order for anything to happen during the simulation. Complicated source mechanisms can be described by having many source commands in the input file. An example with one source command is:

```
source x=5000 y=4000 z=600 m0=1e15 mxx=1 myy=1 mzz=1 \
      type=RickerInt t0=1 freq=5
```

which specifies an isotropic source (explosion) at the point $\mathbf{r}_0 = (5000, 4000, 600)$ with amplitude 10^{15} Nm, using the `RickerInt` time function with offset time $t_0 = 1$ s and frequency parameter $\omega = 5$ Hz. This command sets the off-diagonal moment tensor elements (M_{xy} , M_{xz} and M_{yz}) to zero (which is the default value).

Note that it is not necessary to place the sources exactly on grid points. The discretization of the source terms is second order accurate for any location within the computational domain.

4.2 Predefined time functions

The source time function can be selected from a set of predefined functions described below. All functions start from zero ($\lim_{t \rightarrow -\infty} g(t, t_0, \omega) = 0$) and tend to a constant terminal value,

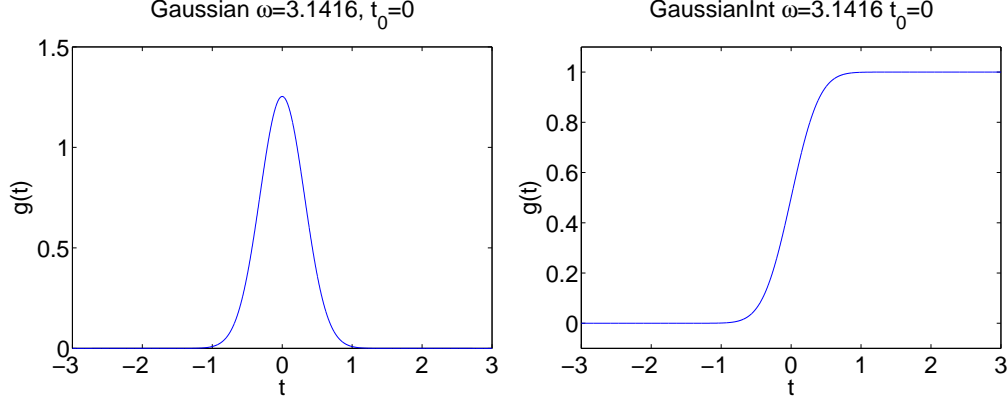


Figure 4.1: Gaussian (left) and GaussianInt (right) with $\omega = \pi$ and $t_0 = 0$.

$\lim_{t \rightarrow \infty} g(t, t_0, \omega) = g_\infty$. In seismic applications, $g_\infty \neq 0$ always corresponds to solving for the displacements of the motion, because the solution will tend to a non-zero steady state solution for large times. This solution corresponds to the final displacements due to a seismic event. When $g_\infty = 0$, the solution will always tend to zero for large times, as is expected from the velocities or accelerations of the motion due to a seismic event.

The Gaussian and the Triangle functions integrate to one ($\int_{-\infty}^{\infty} g(t, t_0, \omega) dt = 1$), while the Sawtooth, Smoothwave, and Ricker functions integrate to zero and have maximum amplitude one. The RickerInt function is the time-integral of the Ricker function and integrates to zero. The GaussianInt, Brune, BruneSmoothed, and Liu functions tend to one ($\lim_{t \rightarrow \infty} g(t, t_0, \omega) = 1$).

The Triangle, Sawtooth, Ramp, Smoothwave, Brune, BruneSmoothed, Liu and VerySmooth-Bump functions are identically zero for $t < t_0$, so they will give reasonable simulation results if $t_0 \geq 0$. However, the Gaussian, GaussianInt, Ricker, and RickerInt functions are centered around $t = t_0$ with exponentially decaying tails for $t < t_0$. Hence t_0 must be positive and of the order $\mathcal{O}(1/\omega)$ to avoid incompatibility problems with the initial conditions. We recommend choosing t_0 such that $g(0, t_0, \omega) \leq 10^{-8}$ for these functions.

4.2.1 Gaussian

$$g(t, t_0, \omega) = \frac{\omega}{\sqrt{2\pi}} e^{-\omega^2(t-t_0)^2/2}.$$

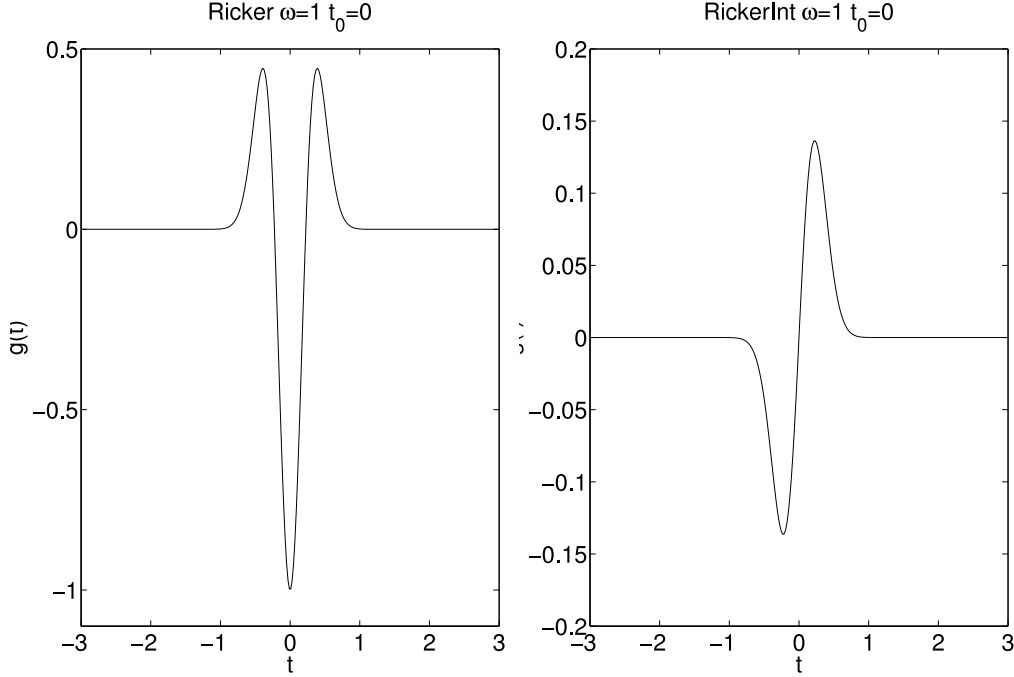
Note that the spread of the Gaussian function (often denoted σ) is related to ω by $\sigma = 1/\omega$. A plot of the Gaussian time-function is shown in Figure 4.1.

4.2.2 GaussianInt

The GaussianInt function is often used in earthquake modeling since it leads to a permanent displacement.

$$g(t, t_0, \omega) = \frac{\omega}{\sqrt{2\pi}} \int_{-\infty}^t e^{-\omega^2(\tau-t_0)^2/2} d\tau.$$

GaussianInt is the time-integral of the Gaussian. A plot of the GaussianInt time-function is shown in Figure 4.1.

Figure 4.2: Ricker (left) and RickerInt (right) with $\omega = 1$ and $t_0 = 0$.

4.2.3 Ricker

$$g(t, t_0, \omega) = (2\pi^2\omega^2(t - t_0)^2 - 1) e^{-\pi^2\omega^2(t-t_0)^2}.$$

A plot of the Ricker time-function is shown in Figure 4.2.

4.2.4 RickerInt

$$g(t, t_0, \omega) = (t - t_0)e^{-\pi^2\omega^2(t-t_0)^2}.$$

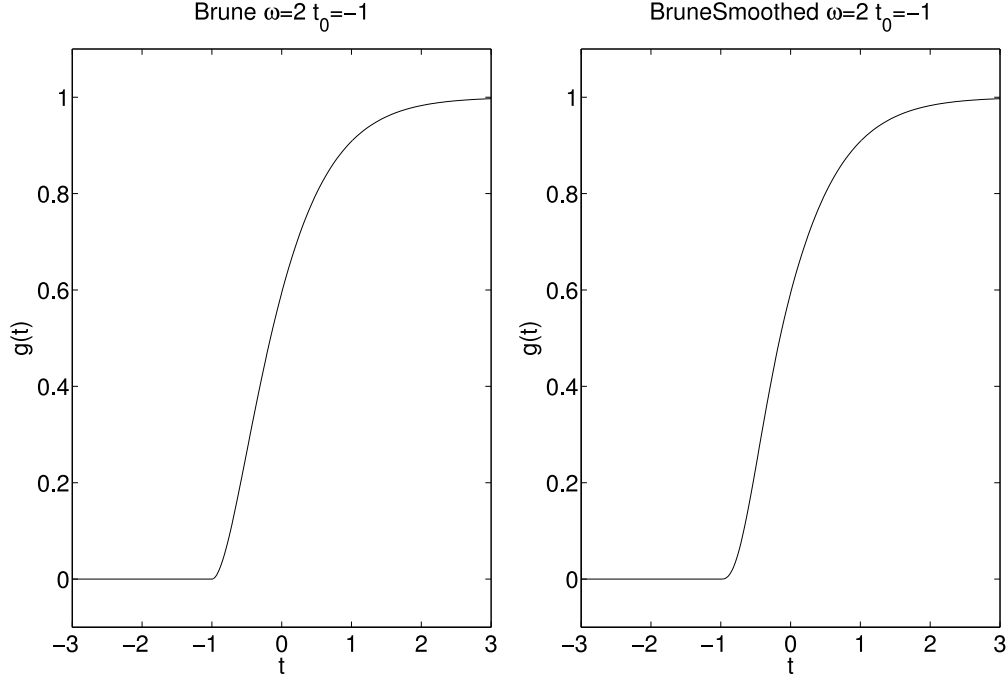
RickerInt is the time integral of the Ricker function, and is proportional to the time-derivative of the Gaussian function. The RickerInt function is sometimes used in seismic exploration simulations. Since the RickerInt function tends to zero for large times, it does not lead to any permanent displacements. A plot of the RickerInt time-function is shown in Figure 4.2.

4.2.5 Brune

The Brune function has one continuous derivative but its second derivative is discontinuous at $t = t_0$,

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 1 - e^{-\omega(t-t_0)}(1 + \omega(t-t_0)), & t \geq t_0. \end{cases}$$

The Brune function is often used in earthquake modeling.

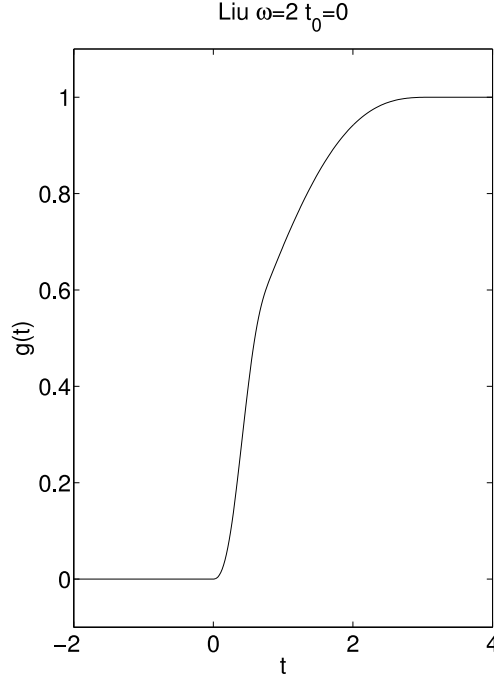
Figure 4.3: Brune (left) and BruneSmoothed (right) with $\omega = 2$ and $t_0 = -1$.

4.2.6 BruneSmoothed

The BruneSmoothed function has three continuous derivatives at $t = t_0$, but is otherwise close to the Brune function:

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 1 - e^{-\omega(t-t_0)} \left[1 + \omega(t-t_0) + \frac{1}{2}(\omega(t-t_0))^2 - \frac{3}{2x_0}(\omega(t-t_0))^3 + \frac{3}{2x_0^2}(\omega(t-t_0))^4 - \frac{1}{2x_0^3}(\omega(t-t_0))^5 \right], & 0 < \omega(t-t_0) < x_0, \\ 1 - e^{-\omega(t-t_0)}(1 + \omega(t-t_0)), & \omega(t-t_0) > x_0. \end{cases}$$

The parameter is fixed to $x_0 = 2.31$. Plots of the Brune and BruneSmoothed time-functions are shown in Figure 4.3. Since the BruneSmoothed function has three continuous derivatives, it generates less high frequency noise. Compared to the Brune function, the BruneSmoothed function gives better accuracy at a given grid resolution

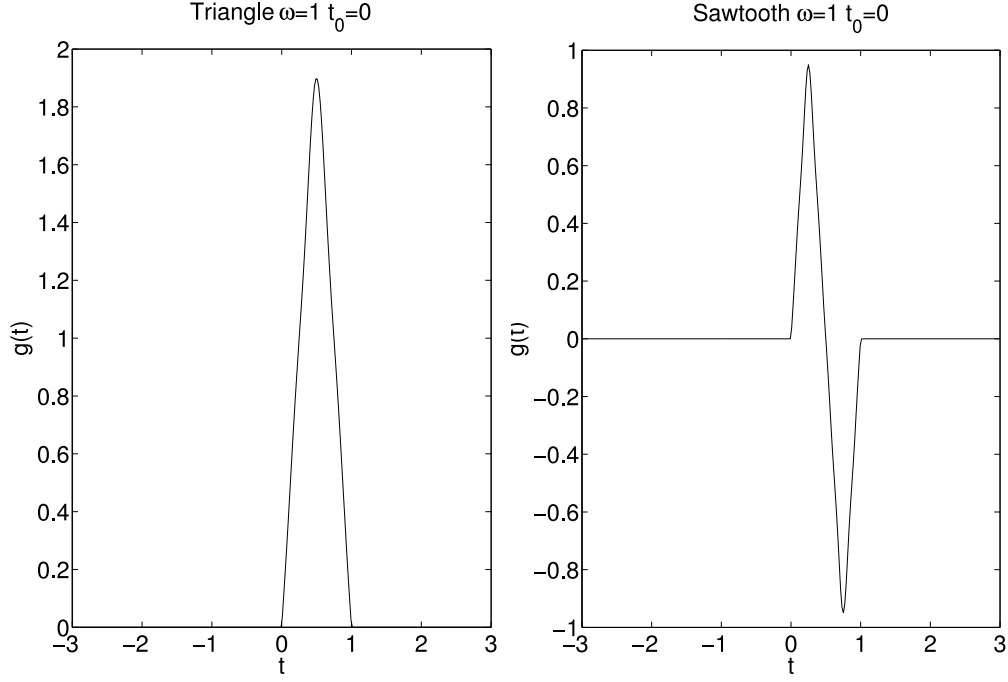
Figure 4.4: Liu time function with $\omega = 2$ and $t_0 = 0$.

4.2.7 Liu

This function was given in a paper by Liu et al., [10]. It is defined by

$$g(t, t_0, \omega) = \begin{cases} 0, & t \leq t_0, \\ C \left[0.7(t - t_0) + \frac{1.2}{\pi}\tau_1 - \frac{1.2}{\pi}\tau_1 \cos\left(\frac{\pi(t - t_0)}{2\tau_1}\right) - \frac{0.7}{\pi}\tau_1 \sin\left(\frac{\pi(t - t_0)}{\tau_1}\right) \right], & t_0 < t \leq \tau_1 + t_0, \\ C \left[t - t_0 - 0.3\tau_1 + \frac{1.2}{\pi}\tau_1 - \frac{0.7}{\pi}\tau_1 \sin\left(\frac{\pi(t - t_0)}{\tau_1}\right) + \frac{0.3}{\pi}\tau_2 \sin\left(\frac{\pi(t - t_0 - \tau_1)}{\tau_2}\right) \right], & \tau_1 + t_0 < t \leq 2\tau_1 + t_0, \\ C \left[0.3(t - t_0) + 1.1\tau_1 + \frac{1.2}{\pi}\tau_1 + \frac{0.3}{\pi}\tau_2 \sin\left(\frac{\pi(t - t_0 - \tau_1)}{\tau_2}\right) \right], & 2\tau_1 + t_0 < t \leq \tau + t_0, \\ 1, & t > \tau + t_0. \end{cases}$$

The parameters are given by $\tau = 2\pi/\omega$, $\tau_1 = 0.13\tau$, $\tau_2 = \tau - \tau_1$, and $C = \pi/(1.4\tau_1\pi + 1.2\tau_1 + 0.3\tau_2\pi)$. The Liu function resembles the Brune function, but the rise is somewhat steeper for small $t - t_0$, see Figure 4.4.

Figure 4.5: Triangle (left) and Sawtooth (right) with $\omega = 1$ and $t_0 = 0$.

4.2.8 Triangle

For $t_0 < t < t_0 + 1/\omega$,

$$g(t, t_0, \omega) = \frac{16\omega}{\pi^2} \left[\sin(\pi\omega(t - t_0)) - \frac{\sin(3\pi\omega(t - t_0))}{9} + \frac{\sin(5\pi\omega(t - t_0))}{25} - \frac{\sin(7\pi\omega(t - t_0))}{49} \right],$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Triangle time-function is shown in Figure 4.5.

4.2.9 Sawtooth

For $t_0 < t < t_0 + 1/\omega$,

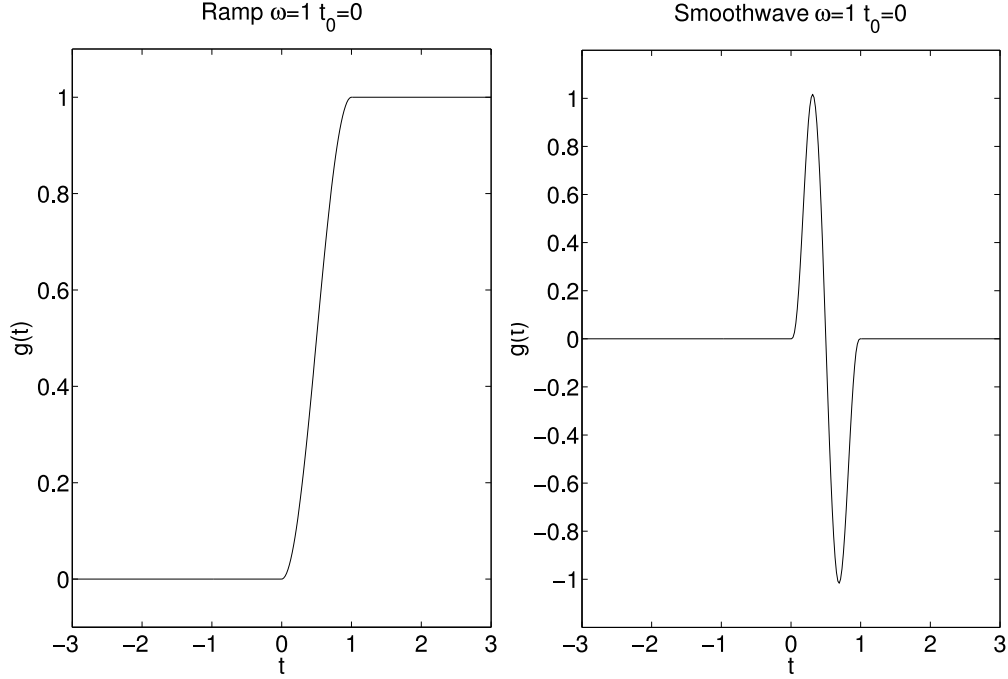
$$g(t, t_0, \omega) = \frac{8}{\pi^2} \left[\sin(2\pi\omega(t - t_0)) - \frac{\sin(6\pi\omega(t - t_0))}{9} + \frac{\sin(10\pi\omega(t - t_0))}{25} - \frac{\sin(14\pi\omega(t - t_0))}{49} \right],$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Sawtooth time-function is shown in Figure 4.5.

4.2.10 Ramp

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 0.5(1 - \cos(\pi(t - t_0)\omega)), & t_0 \leq t \leq t_0 + 1/\omega, \\ 1, & t > t_0 + 1/\omega. \end{cases}$$

A plot of the Ramp time-function is shown in Figure 4.6.

Figure 4.6: Ramp (left) and Smoothwave (right) with $\omega = 1$ and $t_0 = 0$.

4.2.11 Smoothwave

For $t_0 < t < t_0 + 1/\omega$,

$$g(t, t_0, \omega) = \frac{2187}{8}(\omega(t - t_0))^3 - \frac{10935}{8}(\omega(t - t_0))^4 + \frac{19683}{8}(\omega(t - t_0))^5 - \frac{15309}{8}(\omega(t - t_0))^6 + \frac{2187}{4}(\omega(t - t_0))^7,$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Smoothwave time-function is shown in Figure 4.6.

4.2.12 VerySmoothBump

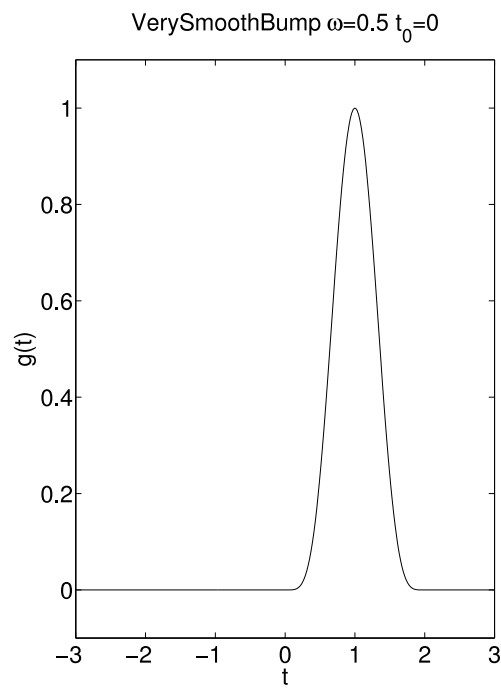
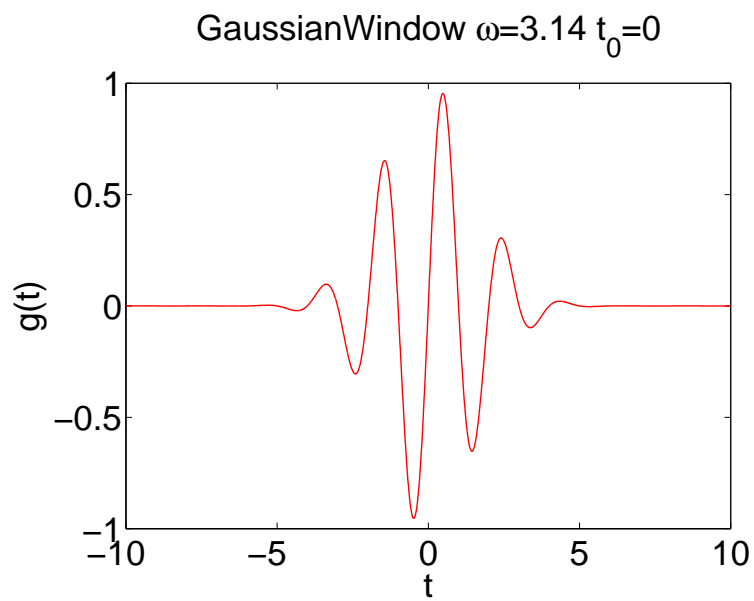
$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ -1024(\omega(t - t_0))^{10} + 5120(\omega(t - t_0))^9 - 10240(\omega(t - t_0))^8 + 10240(\omega(t - t_0))^7 - 5120(\omega(t - t_0))^6 + 1024(\omega(t - t_0))^5, & t_0 \leq t \leq t_0 + 1/\omega, \\ 0, & t > t_0 + 1/\omega. \end{cases}$$

A plot of the VerySmoothBump time-function is shown in Figure 4.7.

4.2.13 GaussianWindow

$$g(t, t_0, \omega) = \sin(\omega t) e^{-(\omega(t-t_0)/N_c)^2/2}$$

A plot of the GaussianWindow time-function with $N_c = 5$ is shown in Figure 4.8. Note that N_c is specified with the `ncyc` keyword, which must be given when this time function is used in the `source` command.

Figure 4.7: VerySmoothBump with $\omega = 0.5$ and $t_0 = 0$.Figure 4.8: GaussianWindow with $\omega = 3.14$, $t_0 = 0$, and $N_c = 5$.

4.3 How fine does the grid need to be?

When preparing the input file, the most difficult parameter to choose is probably the grid size h . It is extremely important to use a grid size which is sufficiently small, because you must resolve the waves that are generated by the source. On the other hand you don't want to use an unnecessarily small grid size, because both the execution time and the memory requirements increase with decreasing grid size, see Section 13.

The number of grid points per shortest wavelength, P , is a normalized measure of how well a solution is resolved on the computational grid. Since the shear waves have the lowest velocities and a shorter wave length than the compressional waves, the shortest wave length L_{min} can be estimated by

$$L_{min} = \frac{\min V_s}{f_{max}},$$

where V_s is the shear velocity of the material and f_{max} is the largest significant frequency in the time function $g(t)$. Hence the number of grid points per wave length equals L_{min}/h , which is given by

$$P = \frac{\min V_s}{h f_{max}}. \quad (4.1)$$

Note that h needs to be made smaller to maintain the same value of P if either V_s is decreased or if the frequency is increased. In formula (4.1), $\min V_s$ is found from the material properties and h is determined by the input grid specification. The frequency spectrum of the solution is determined by the frequency spectrum of the time function(s) in the source term(s).

Figure 4.9 displays the absolute values of the Fourier transforms of the functions Gaussian, RickerInt, Ricker, and the time derivative of the Brune function. Inspection of the mathematical definitions of the Gaussian and Brune functions shows that the **freq** parameter specifies the angular frequency for these functions. The relation between the fundamental frequency f_0 and the **freq** parameter is given by

$$f_0 = \begin{cases} \mathbf{freq}, & \text{for Ricker, RickerInt, and VerySmoothBump,} \\ \frac{\mathbf{freq}}{2\pi}, & \text{for Liu, Brune, BruneSmoothed, Gaussian, GaussianInt, and GaussianWindow.} \end{cases} \quad (4.2)$$

The plots in Figure 4.9 were made with frequency parameter **freq**=0.25 for the Ricker and RickerInt functions and frequency parameter **freq**=1.5 for the Gaussian and $d/dt(\text{Brune})$ functions. Hence, $f_0 \approx 0.25$ for all functions in Figure 4.9. Note that the Fourier transform of the Brune function decays much slower than the other functions for high frequencies. This is due to its lack of smoothness at $t = t_0$.

It is the upper power (highest significant) frequency in the time function that shall be used in (4.1) to estimate the number of grid points per wave length. For practical purposes f_{max} can be defined as the frequency where the amplitude of the Fourier transform falls below 5 % of its max value. We have

$$f_{max} \approx \begin{cases} 2.5f_0, & \text{Ricker, RickerInt, Gaussian time functions,} \\ 4f_0, & \text{Brune time function.} \end{cases} \quad (4.3)$$

In other words, simulations using the Brune function are harder to resolve on the grid and need much more grid points to give reliable results.

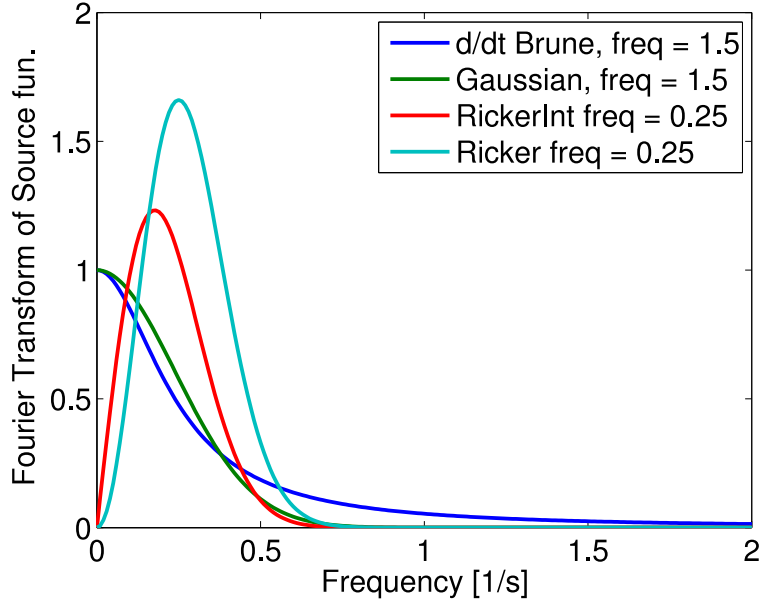


Figure 4.9: Magnitude of the Fourier transform of the derivative of the Brune (dark blue), the Gaussian (green), the RickerInt (red), and the Ricker (light blue) time-functions. Here `freq`=1.5 for the Gaussian and the derivative of the Brune function, and `freq`=0.25 for Ricker and RickerInt.

Our experience is that *WPP* gives accurate results for

$$P \geq 15,$$

but the exact number depends on the distance between the source and the receiver. Note that the relation between the fundamental frequency f_0 and the upper power frequency f_{max} in (4.3) is very important. For other time functions, f_{max} can be estimated using the matlab/octave scripts `fcnplot` and `ftfcnplot` in the `tools` directory. A lower number for P can be used in some practical situations. The best way of checking the accuracy in a numerical solution is to repeat the calculation on a finer mesh, even though this approach can become computationally expensive.

We finally remark that the ratio between the compressional and shear velocity, V_P/V_S , can have a significant influence on the accuracy of surface waves, see the paper by Kreiss and Petersson [8] for details.

4.3.1 Lamb's problem

We now compute solutions to Lamb's problem in a material with $V_p = \sqrt{3} \text{ km/s}$, $V_s = 1 \text{ km/s}$ and the density 1000 kg/m^3 . The solution is forced downward with amplitude `fz=5e13 N` and with a time function centered at time `t0=25 s`. For various time functions the solution is recorded at receivers 10 and 50 km from the source. At the receivers the relative error

$$\frac{\|u_{exact}(t) - u_{computed}(t)\|_{\infty}}{\|u_{exact}(t)\|_{\infty}},$$

in the horizontal component is computed and plotted in Figure 4.10. In these calculations, the grid size was held constant and the frequency parameter `freq` was varied. Note that the reported

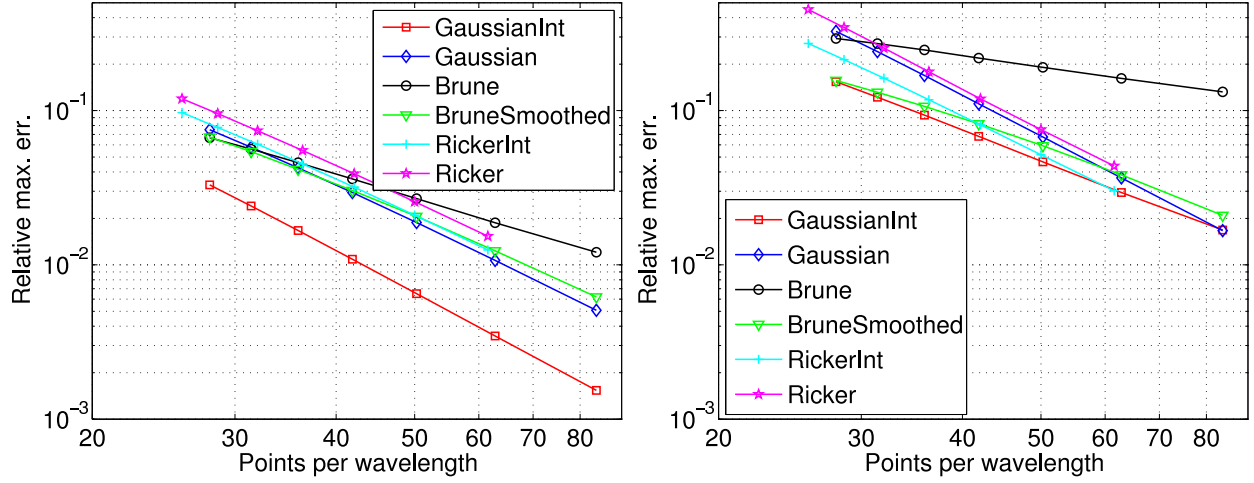


Figure 4.10: Relative errors for different source functions 10 (left) and 50 km (right) from the source. For the Brune time function the error decays much slower than for the other time functions. Here, the number of points per wavelength (P) was based on the fundamental frequency f_0 instead of f_{max} , so the values of P should be divided by 2.5 for GaussianInt, Gaussian, RickerInt, and Ricker. For Brune, P should be divided by 4.

number of points per wavelength was based on the fundamental frequency f_0 instead of f_{max} , so the values of P should be reduced according to (4.3)

From Figure 4.10 we see that for all of the time functions, except the Brune function, there is a decrease in error inversely proportional to the square of the number of points per wavelength. The errors are larger for the Brune function since its spectrum decays much slower due to its discontinuous second derivative at $t = t_0$. The difference in the error levels between the left and the right sub-figures are due to the fact that errors in the numerical solution accumulate as the solution propagates away from the source. For a single harmonic wave, and a second order accurate finite difference method, the number of points per wavelength required to achieve a certain error is proportional to the square root of the number of wavelengths the wave propagates (see Chapter 3 in [7] for a detailed discussion). Thus, to get the same accuracy at five times the distance from the source, we need to use about $\sqrt{5} \approx 2.24$ times more points per wave length. This could be achieved by reducing the grid size by a factor 2.24 in each direction, resulting in a factor of 11.1 times more grid points and an increase in CPU time by a factor 25.

Chapter 5

The material model

The elastic material model in *WPP* is defined by the values of the density, ρ , the compressional velocity, V_p , and the shear velocity, V_s , at each grid point. These values can be specified by the block command (§ 5.1), the efile command (§ 5.2), the pfile command (§ 5.3), the ifile command (§ 5.4), or by a combination of them. It is also possible to use a visco-elastic material model. This is described in chapter 8.

Note that *WPP* uses a single layer of ghost points outside the computational domain (as defined by the grid command). The material properties must therefore be defined for the computational domain padded by one layer of ghost points. Note, however, that the material model does not need to be defined above the free surface. Material properties at those ghost points are instead assigned by extrapolation from the interior of the domain.

It is important to note that the order within the material commands (block, pfile, efile, and ifile) *does* matter (unlike all other commands) in that the priority of the material command increases towards the end of the input file. Hence, a material command in the input file can be completely or partially overridden by subsequent material commands.

In the block, pfile, and ifile commands, material properties are assigned based on the depth below the free surface. This means that the internal material model depends on the topography, but the material properties along the free surface will always be the same, independent of the elevation of the topography model. For the efile command, material properties are defined as functions of elevation relative to mean sea level ($z = 0$). Here the topography information is embedded in the material description. If you combine the efile command with a planar topography, a linear mapping is constructed before the material properties are assigned. In the top (finest) Cartesian grid, the properties at the free surface are thus mapped to the top grid surface ($z = 0$), and the bottom grid surface (with $z = z_N$) is assigned material properties for elevation $-z_N$. Elevation values at intermediate grid points follow from the linear mapping. Subsequent (coarser) Cartesian grids are not effected by this mapping procedure.

After reading all material commands in the input file and assigning material properties to the computational grid, *WPP* outputs general information about the ranges in the material model. For a purely elastic material, the output looks like

```
----- Material properties ranges -----  
1590 kg/m^3 <= Density <= 3300 kg/m^3  
768 m/s      <= Vp      <= 7790 m/s  
500 m/s      <= Vs      <= 4420 m/s
```

```

1.536      <=  Vp/Vs      <=  4.48
3.975e+08 Pa      <=  mu      <=  6.44701e+10 Pa
1.4282e+08 Pa      <=  lambda  <=  7.13863e+10 Pa
-----

```

It is always a good idea to check that these numbers are reasonable before proceeding with the simulation. We also recommend inspecting the material model along a few **image** planes.

5.1 The block command

The block command can be used to specify material properties in rectangular volumes of the computational domain, either with constant values or linear vertical gradients. By combining the block command with the sub-region options we can define a material model composed of three layers:

```

block vp=4000 vs=2500 rho=2000
block vp=6000 vs=3500 rho=2700 z1=15000
block vp=8000 vs=4500 rho=3300 z1=35000 z2=100000

```

In this case the top layer has a thickness of 15 km, the middle layer 20 km and the lower layer 65 km. Because these block commands do not specify horizontal coordinates, the values extend to the grid boundaries in both horizontal directions. To add a box shaped inclusion of a new material we could add the following line

```

block vp=3000 vs=2000 rho=1000 \
  x1=4000 x2=8000 y1=3000 y2=7000 z1=10000 z2=70000

```

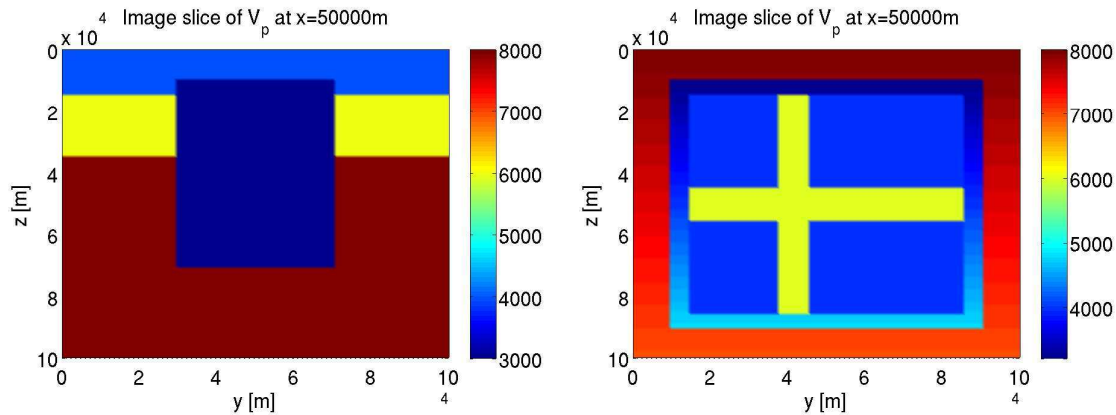


Figure 5.1: Examples of material models specified with the block command.

To the left in Figure 5.1 an image slice of V_p through $x = 50,000$ is displayed.

The following example combines several block commands used to generate the material model displayed to the right in Figure 5.1:

```

block vp=8000 vs=4500 rho=3300 vpgrad=-0.01
block vp=3000 vs=2000 rho=1000 \

```

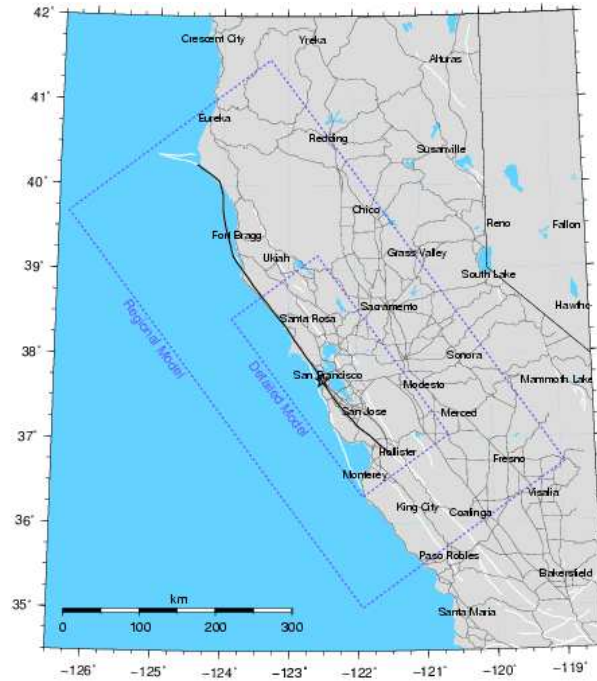


Figure 5.2: The geographical extent of the etree models for Northern California and the San Francisco bay area.

```

x1=1e4 x2=9e4 y1=1e4 y2=9e4 z1=1e4 z2=9e4 vpgrad=0.02
block vp=4000 vs=2500 rho=2000 \
  x1=15e3 x2=85e3 y1=15e3 y2=85e3 z1=15e3 z2=85e3
block vp=6000 vs=3500 rho=2700 \
  x1=15e3 x2=85e3 y1=15e3 y2=85e3 z1=45e3 z2=55e3
block vp=6000 vs=3500 rho=2700 \
  x1=15e3 x2=85e3 z1=15e3 z2=85e3 y1=38e3 y2=45e3

```

5.2 The efile command

The efile command is used to read in material properties from an etree database file. Etree databases use an oct-tree data structure which allows material properties to be represented with finer spatial resolution near the surface. Topography and bathymetry information is included in the database. The same etree database file can be used independently of the grid size, so there is no need to have a one-to-one mapping between the etree model and the computational grid. Unfortunately, it takes a major effort to develop an etree database file, and we currently only have access to material data for Northern California and the extended San Francisco bay area. This model was developed by the USGS and can currently be accessed from <http://earthquake.usgs.gov/regional/nca/3Dgeologic>. Be aware that the database is rather large and can take a very long time to download. The geographical extent of the etree model is given in Table 5.1, which also is shown on a map in Figure 5.2.

In the etree database, material properties are stored as functions of geographical coordinates

Detailed Model			Regional Model		
Corner	Longitude	Latitude	Corner	Longitude	Latitude
SE	-120.64040	37.04718	SE	-118.944514	36.702176
SW	-121.91833	36.31746	SW	-121.930857	35.009018
NW	-123.85736	38.42426	NW	-126.353173	39.680558
NE	-122.56127	39.17461	NE	-123.273199	41.48486

Table 5.1: Geographical extent (NAD27 projection) for the central California velocity models. Both models are defined down to 45 km depth. See <http://www.sf06simulation.org/geology/velocitymodel> for details.

(latitude, longitude, elevation). *WPP* uses formulas (3.4)-(3.5) to determine the geographical coordinates for each grid point before it obtains the material properties from the data base. Internally to *WPP*, the *cencalvm* software library is used to query the etree database, which in turn relies on additional libraries. Hence, before the *efile* command can be used, the corresponding software libraries must be installed and *WPP* must be configured to use them, see Section A for details.

It is important to note the bounds of the geographical region in the database. Assuming the computational domain is contained within the bounds of the database, it is easy to set up the material model in the input file:

```
grid x=100e3 y=100e3 z=40e3 lat=38.0 lon=-121.8 az=144 h=1000
efile etree=/p/lscratchd/andersp/USGSBayAreaVM-08.3.0.etree
```

To verify that the computational domain is inside the etree data base, we recommend checking the geographical coordinates on a map during the construction of the input file. We often use the Google Earth program for this purpose. In the case when the computational domain is larger than the region covered by the *efile*, a *block* command can be used to assign material properties to grid points outside of the *efile* region:

```
grid x=300000 y=300000 z=60000 lat=38 lon=-121.5 az=135 nx=100
block vp=8000 vs=4000 rho=1000 rhograd=0.5
efile etree=/p/lscratchd/andersp/USGSBayAreaVM-08.3.0.etree
```

However, sharp jumps in material properties can lead to significant scattering of seismic waves. In some cases, better results can be obtained by reducing the size of the computational domain to match the extent of the etree region.

To enable use of the extended SF model, the extended etree file must also be downloaded and then added to the *efile* command line (file names have been shortened for improved readability):

```
efile etree=USGSBayAreaVM.etree xetree=USGSBayAreaVMExt.etree
```

5.3 The pfile command

The *pfile* command can be used to assign material properties based on depth profiles. A *pfile* contains the values of the model features (P-velocity, S-velocity, density, and Q-factors) as function

of depth at points on a regular lattice covering the horizontal extent of the computational domain. The points on the lattice are either defined by their latitude and longitude coordinates, or by the x and y -coordinates. The number of grid points in the depth direction needs to be the same for all profiles, but the grid spacing does not need to be uniform and can also be different for each profile. Material discontinuities can be represented by two material values for the same depth value. Material layers, which only occur in a subset of the profiles, can be tapered to have zero thickness in the remaining profiles. This is handled by introducing multiple data points with the same depth and material values in a profile.

The lattice of the pfile does not need to have any relation to the computational mesh used in *WPP* and is often much coarser. The material properties in the computational mesh are assigned values using Gaussian averaging between the nearest $N_G \times N_G$ profiles in the latitude-longitude plane and linear interpolation in the depth direction. Let the grid point have longitude θ , latitude ϕ and depth d . Material properties are first linearly interpolated in the depth direction along each profile and then averaged in the latitude-longitude plane. The number of points in the Gaussian averaging, N_G , is assigned by the user in the `pfile` command. For example, the following line in the input file makes *WPP* read a pfile named `material.ppm`:

```
pfile filename=material.ppm vmin=1000 vpmin=1732 smoothingsize=4
```

The optional `vmin` and `vpmin` keywords are used to assign minimum threshold values for the P - and S -velocities, respectively. Here, `smoothingsize=4` means that $N_G = 4$ in the Gaussian averaging. A larger value of N_G (≥ 5) is particularly useful to avoid staircasing imprints when the computational grid is much finer than the pfile lattice, see Figure 5.3. The `smoothingsize` keyword can be assigned any number greater than or equal to one.

When N_G is odd, the Gaussian averaging starts by finding the closest grid point on the latitude-longitude lattice, (ϕ_i, θ_j) . The material property c (ρ , V_p , V_s , etc.) is assigned by the formula

$$c(\phi, \theta) = \frac{\sum_{m=i-Q}^{i+Q} \sum_{n=j-Q}^{j+Q} c_{m,n} \omega_{m,n}}{\sum_{m=i-Q}^{i+Q} \sum_{n=j-Q}^{j+Q} \omega_{m,n}}, \quad Q = \frac{N_G - 1}{2}, \quad (5.1)$$

where the weights are given by

$$\omega_{m,n} = e^{-[(\phi_m - \phi)^2 + (\theta_n - \theta)^2] / \alpha^2}, \quad \alpha = \frac{N_G \Delta_{lat}}{2\sqrt{-\log 10^{-6}}},$$

and the grid size in the latitude-longitude lattice is Δ_{lat} . This choice of α makes the weights $\omega_{m,n} < 10^{-6}$ for points that are further from (ϕ_m, θ_n) than $N_G \Delta_{lat} / 2$, which justifies the truncation of the series in (5.1). A similar procedure is used for even values of N_G , but in this case the averaging formula (5.1) is centered around the nearest cell center on the latitude-longitude lattice.

By default the pfile data are assumed to be given in latitude-longitude coordinates. It is also possible to read pfiles that have data given on a Cartesian (x - y) grid. Use the option `style`, as in the example

```
pfile filename=materialxy.ppm vmin=1000 vpmin=1732 smoothingsize=4 \\  
style=cartesian
```

to indicate that the given pfile contains data on a grid of Cartesian coordinates.

Data files for the pfile command are written in an ASCII text format. See Section 12.2 for a description of both the latitude-longitude pfile and the Cartesian pfile grid formats.

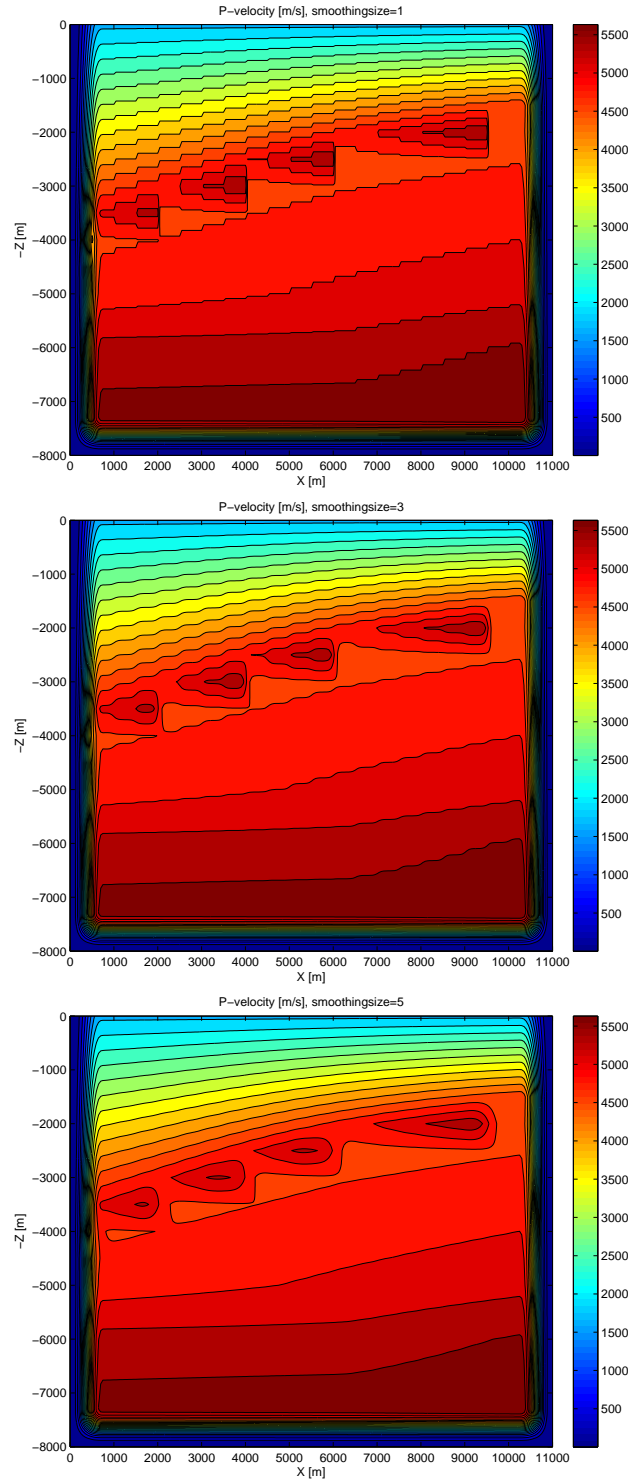


Figure 5.3: The `smoothing size` parameter can be used to average out imprinting from the horizontal lattice in a coarse pfile material model. Here we show V_P in the plane $y = 5,000$ as function of x and $-z$. In this case, `smoothing size`= 1 in the top, `smoothing size`= 3 in the middle, and `smoothing size`= 5 in the bottom plot.

5.4 The ifile command

The **ifile** command reads a file holding the depth to material interface surfaces. The material properties between each pair of material surfaces must be defined by the **material** command. The depth must be non-negative. Zero depth corresponds to the topography. Material surfaces are specified on a regular lattice in geographic coordinates. The unit for depth is meters, while latitude and longitude are in degrees. The **ifile** command may be combined with other material specifications and it is *not* necessary that the lattice in geographical coordinates covers the horizontal extent of the computational domain.

Let $N_{mat} \geq 1$ material surfaces be known at longitudes

$$\phi_i, \quad i = 1, 2, \dots, N_{lon},$$

and latitudes

$$\theta_j, \quad j = 1, 2, \dots, N_{lat},$$

Note that the latitudes and the longitudes must either be strictly increasing or strictly decreasing, but the step size may vary. Also note that the lattice points are independent of those in the **topography** command.

The material surfaces should be given on the regular lattice

$$d_{q,i,j} = \text{depth to surface number } q \text{ at longitude } \phi_i, \text{ latitude } \theta_j.$$

The material surfaces correspond to material properties in the following way. At longitude ϕ_i , latitude θ_j material number 1 (as defined by the **material** command) occupies depths $0 \leq d \leq d_{1,i,j}$. Material number 2 occupies depths $d_{1,i,j} \leq d \leq d_{2,i,j}$, and so on. If $d_{1,i,j} = 0$, material number 1 is not used. Similarly, material number $k > 1$ is not used if $d_{k-1,i,j} = d_{k,i,j}$. Material properties are only defined for depths down to the last surface, i.e.,

$$0 \leq d \leq d_{N_{mat},i,j}.$$

If the computational domain extends below the last material surface, it is necessary to use other commands to define the material properties in those regions.

The material properties can have a constant, linear, quadratic, and square root dependence of depth. For example, the most general dependence for density is

$$\rho(d) = \rho_{k,0} + \rho_{k,1}d + \rho_{k,2}d^2 + \rho_{k,1/2}\sqrt{d}, \quad d_{k-1,i,j} \leq d < d_{k,i,j}.$$

Bi-linear interpolation in longitude and latitude is used to define the material surfaces in between the data points. Note that only constant values are supported for the quality factors (Q_P and Q_S) within each material.

An example that uses an ifile material description is discussed in Section 10.3. The **ifile** file format is described in Section 12.3.

Chapter 6

Topography

The topography command in *WPP* is used to specify the shape of the top surface of the computational domain,

$$z = \tau(x, y).$$

Three different topography descriptions are currently implemented in *WPP*: a Gaussian hill (§ 6.1), a latitude-longitude grid file (§ 6.2), or topography from an Etree data base (§ 6.3).

A curvilinear grid is automatically constructed between the topography surface and a user specified depth $z = \mathbf{zmax}$. If no topography command is present in the input file, the top surface is taken to be the plane $z = 0$, and no curvilinear grid is constructed. If the topography surface $z = \tau(x, y)$ varies between $\tau_{min} \leq z \leq \tau_{max}$ (z is positive downwards), the grid generation usually works well if

$$\mathbf{zmax} \geq \tau_{max} + 2(\tau_{max} - \tau_{min}), \quad (6.1)$$

After reading the topography, *WPP* prints out the min and max z -coordinates, as well as the specified value of \mathbf{zmax} ,

```
***Topography grid: min z = -1.1443e+03, max z = 1.0929e+03, \  
top Cartesian z = 6.000000e+03
```

In this case, which corresponds to the topography shown in Figure 6.1,

$$\tau_{min} = -1144.3, \quad \tau_{max} = 1092.9,$$

and $\mathbf{zmax} = 6000$. We have $\tau_{max} + 2(\tau_{max} - \tau_{min}) = 5567.3$, which satisfies (6.1).

Except for the Gaussian hill topography, the topography surface is smoothed by a Jacobi iteration before the curvilinear grid is generated,. The purpose of the smoothing is to ensure that the variations in topography can be resolved on the computational grid. By default, 10 iterations are performed and this gives a satisfactory result in many cases. It is possible to change the number of iteration by using the `smooth` option in the topography command. You can inspect the result of the smoothing by saving the top grid surface in an image file,

```
image mode=grid z=0 cycle=0 file=test
```

Note that the z coordinate (positive downwards) is saved on a grid image file, while the elevation (positive upwards) of the raw (before smoothing) topography is saved on a topo image file.

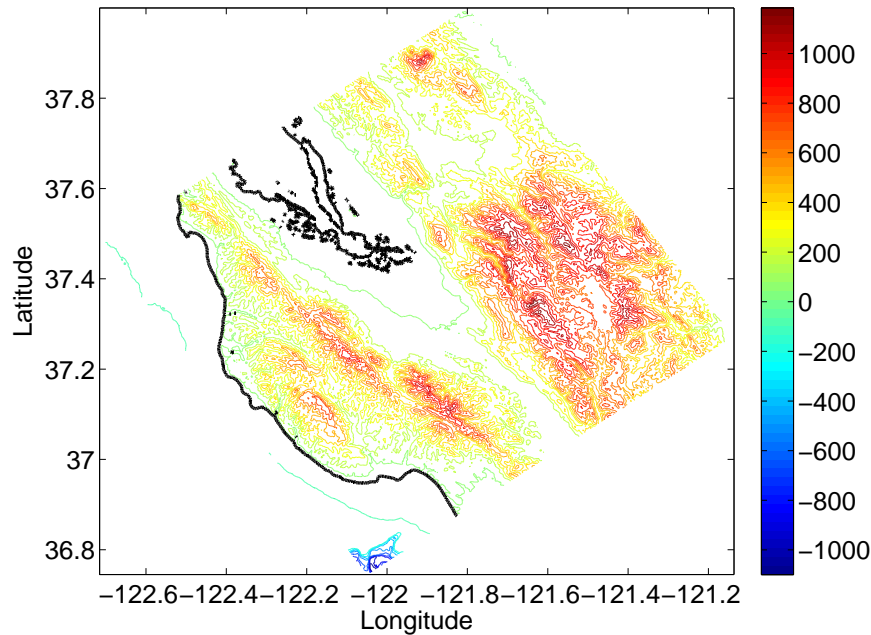


Figure 6.1: Topography and bathymetry in the vicinity of San Jose, south of San Francisco. The coastline is outlined by a thicker black line. Note the deep water in the Monterey Canyon, near the bottom corner of the computational domain.

6.1 Gaussian hill topography

The simplest type of topography is a Gaussian hill, which allows the user to place one Gaussian hill at a specified location in the (x, y) -plane. The user can adjust the amplitude of the hill as well as its spread in the x and y -directions. The Gaussian hill topography command has the following syntax:

```
topography input=gaussian zmax=7.5 gaussianAmp=2.4 \
    gaussianXc=3.6 gaussianYc=2.4 \
    gaussianLx=0.25 gaussianLy=0.3
```

Note the `zmax` option, which tells *WPP* to extend the curvilinear grid to $z = 7.5$. The most common use of the Gaussian hill topography is for testing, see for example the input scripts in `examples/twilight`:

```
gauss-twi-1.in gauss-twi-2.in gauss-twi-3.in
```

6.2 Topography grid file

The topography can be given on a regular lattice in geographical (lat-lon) coordinates. This approach works well together with the `block`, `pfile`, and `ifile` material commands. When the material is described by an `efile` command, it is better to setup the topography from the same etree database, see Section 6.3.

To setup the topography for the Grenoble basin test case described in Section 10.3, you give the command

```
topography input=grid file=grenobleCoarse.topo zmax=3000 order=2
```

The file `grenobleCoarse.topo` holds the elevation (in meters) relative to mean sea level and must conform to the simple ASCII text format described in Section 12.3. In the above case, a curvilinear grid is constructed between the topography surface and $z = 3000$, and the `order=2` option specifies a second order polynomial stretching in the curvilinear mapping function. The topography is shown in Figure 6.2.

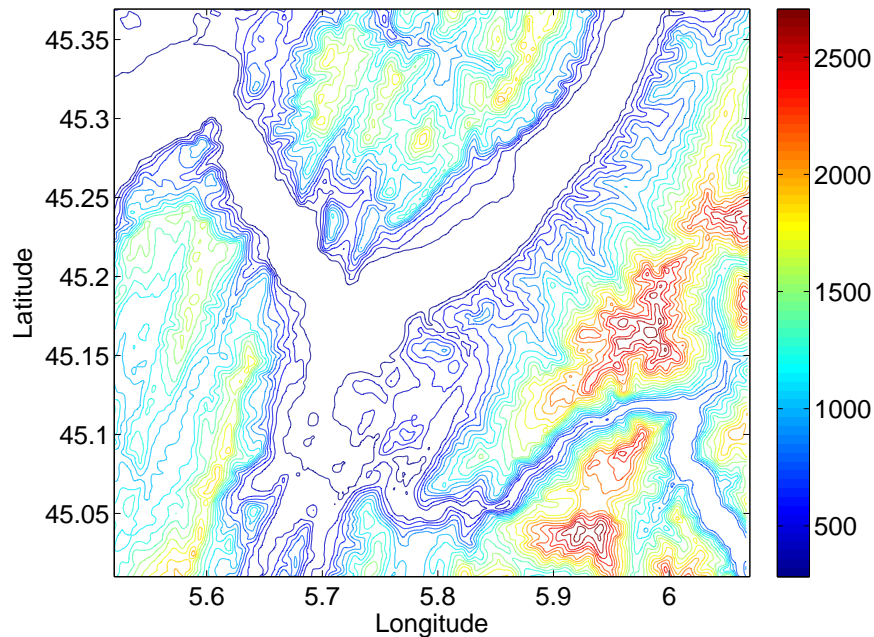


Figure 6.2: Topography in the vicinity of Grenoble, France.

6.3 Etree topography

The Etree data bases for the San Francisco bay area and Northern California contain topographic information. You can setup the computational grid to follow this topography by using the commands

```
topography input=efile zmax=6e3 order=2
efile query=MAXRES \
    etree=/Users/petersson1/src/wpp/tests/USGSBayAreaVM-08.3.0.etree
```

Here, the topography command tells *WPP* to read the topography from the Etree specified by the `efile` command. Hence, the topography command must be accompanied by an `efile` command. The `order=2` option specifies the type of stretching to use when making the curvilinear grid. A

higher value makes the curvilinear grid smoother near the bottom, but can cause a larger variation in grid size near the top. The `zmax=6e3` option tells *WPP* to extend the curvilinear grid down to $z = 6000$.

Chapter 7

Mesh refinement

The refinement command in *WPP* enables the user to locally refine the computational mesh in areas where finer resolution is needed, i.e., where the wave speed is small. In order to maintain a constant resolution in terms of the number of grid points per wavelength for a given frequency (see Equation (4.1)), the grid size should be adjusted such that ratio V_s/h becomes approximately constant over the computational domain. In *WPP*, we use a composite grid approach consisting of a set of structured component grids with hanging nodes on the grid refinement interfaces. This allows the grid resolution to follow the main variations in wave speed, and gives ideal wave propagation properties in each component grid. To guarantee stability of the numerical scheme, an energy conserving coupling approach is used to couple the solution across grid refinement interfaces, see [13] for details.

When using mesh refinement, the extent of the computational domain is determined by the grid command, which also specifies the grid size in the coarsest component grid,

```
grid h=2000 x=40000 y=40000 z=40000
```

The two refinement commands

```
refinement zmax=30000  
refinement zmax=2000
```

specify two mesh refinement interfaces: $z_1 = 30000$, and $z_2 = 2000$. As a result, the composite grid contains three component grids, where the coarsest component has grid size $h = 2000$ and covers the bottom of the computational domain: $z_1 \leq z \leq 40000$. Next refinement grid has half the grid size ($h = 1000$) and covers $z_1 \leq z \leq z_2$. The grid size in the third component is another factor of two smaller ($h = 500$) and covers the top of the computational domain: $z_2 \leq z \leq 0$. The composite grid is shown in Figure 7.1, where the grid is plotted in the vertical $x = 20000$ plane. Note that refinement grids are aligned in the sense that every second grid point coincides with a grid point in the next coarser grid.

Mesh refinement can also be used together with topography. Here we use an example from the Alum Rock simulation described in 10.4. The composite grid is setup with the commands

```
grid x=100e3 y=100e3 z=40e3 lat=38.0 lon=-121.8 az=144 h=1000  
refinement zmax=10e3  
refinement zmax=7000  
topography input=efile zmax=6e3 order=2
```

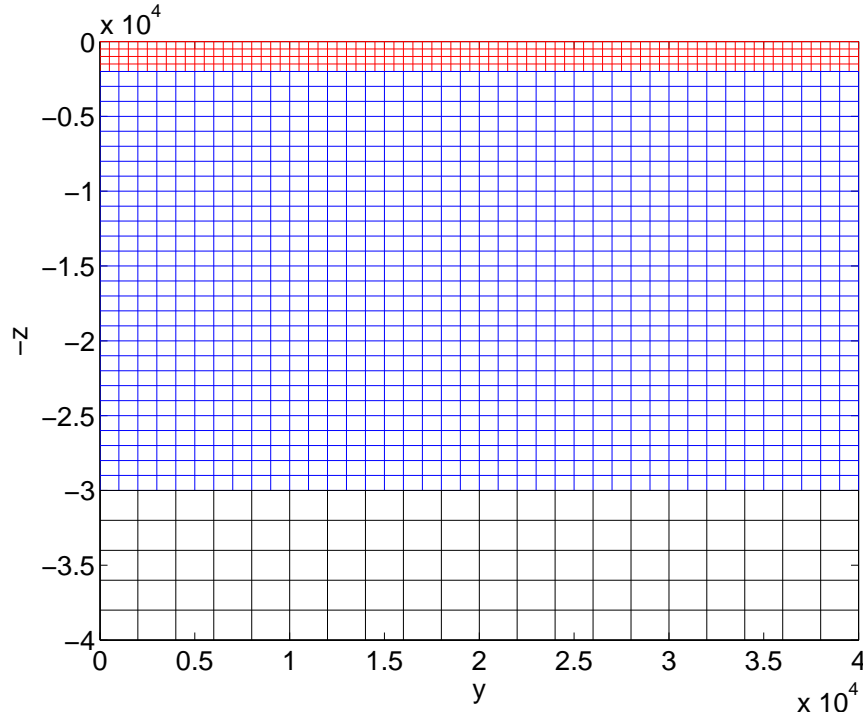


Figure 7.1: Composite grid with two mesh refinement interfaces and three Cartesian component grids.

```
efile query=MAXRES vsmin=500 vpmin=768 \
  etree=/p/lscratchd/andersp/USGSBayAreaVM-08.3.0.etree
```

Here the base grid, which is always Cartesian, has grid size $h = 1000$ and covers $10000 \leq z \leq 40000$. Next Cartesian grid has half the grid size ($h = 500$) and covers $7000 \leq z \leq 10000$. The grid size in the finest Cartesian component grid is reduced by another factor of two, which gives $h = 250$. This component extends to the bottom of the curvilinear grid, i.e., $7000 \leq z \leq 6000$. The vertical extent of the curvilinear grid is specified by the `zmax=6000` option in the topography command, i.e., the curvilinear grid covers the domain between $z = 6000$ and the topography surface, $z = \tau(x, y)$. In the horizontal directions, the grid size in the curvilinear grid is the same as in the finest Cartesian grid. The number of grid points in the vertical direction is chosen such that the average vertical grid size is the same as the grid size in the horizontal directions. A portion of the computational grid is shown in the vertical cross section $x = 50000$, see Figure 7.2.

After constructing the computational grid, *WPP* outputs information about the number of grid points in each component grid. For the above example, we get

Global grid sizes (without ghost points)

Grid	h	Nx	Ny	Nz	Points
0	1000	101	101	31	316231
1	500	201	201	7	282807
2	250	401	401	5	804005
3	250	401	401	25	4020025

Total number of grid points (without ghost points): 5.42307e+06

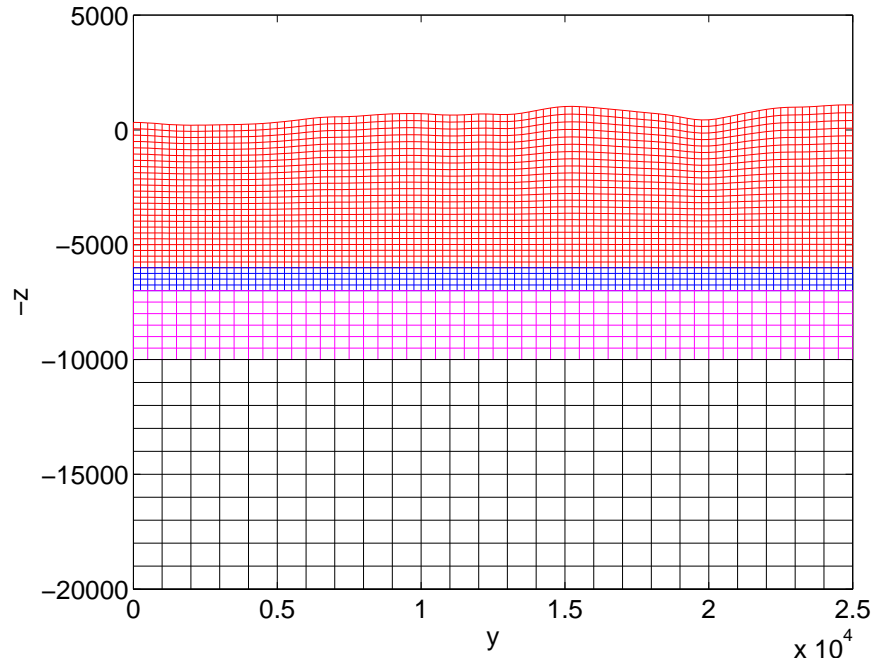


Figure 7.2: A composite grid with two mesh refinement interfaces and topography. In this case there are three Cartesian components and one curvilinear grid following a non-planar topography.

Note that most grid points are in grids number 2 and 3, with grid size $h = 250$. A little further down in the output file, *WPP*, provides information about the resolution in terms of grid points per wave length:

```
***** PPW = minVs/h/maxFrequency *****
g=0, h=1.000000e+03, minVs/h=3.29 (Cartesian)
g=1, h=5.000000e+02, minVs/h=6.16 (Cartesian)
g=2, h=2.500000e+02, minVs/h=12.04 (Cartesian)
g=3, h=2.500000e+02, minVs/h=2 (curvilinear)
```

As is common in seismic applications, the material velocities are the lowest near the free surface, i.e., in grid number 3 in this case. From this information, we can estimate the highest frequency that can be reliably propagated on this mesh. To get $P = 15$ grid points per wave length, we can use a maximum frequency of

$$f_{max} = \min \frac{V_S}{hP} = \frac{2}{15} \approx 0.13\text{Hz}.$$

Chapter 8

Attenuation

8.1 Viscoelastic modeling

WPP implements a linear viscoelastic material model by superimposing n standard linear solid (SLS) mechanisms, leading to the governing equations

$$\rho \frac{\partial^2 \mathbf{u}}{\partial t^2} = \mathbf{L}(\lambda_0, \mu_0) \mathbf{u} - \sum_{\nu=1}^n \mathbf{L}(\lambda_\nu, \mu_\nu) \bar{\mathbf{u}}^{(\nu)} + \mathbf{F}, \quad \mathbf{x} \in \Omega, \quad t \geq 0, \quad (8.1)$$

where the spatial operator is

$$\mathbf{L}(\lambda, \mu) \mathbf{u} =: \nabla(\lambda(\nabla \cdot \mathbf{u})) + \nabla \cdot (2\mu \mathcal{D}(\mathbf{u})), \quad \mathcal{D}(\mathbf{u}) = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T). \quad (8.2)$$

The memory variables, $\bar{\mathbf{u}}^{(\nu)}$, in (8.1) are governed by the differential equations

$$\frac{1}{\omega_\nu} \frac{\partial \bar{\mathbf{u}}^{(\nu)}}{\partial t} + \bar{\mathbf{u}}^{(\nu)} = \mathbf{u}, \quad \mathbf{x} \in \Omega, \quad t \geq 0, \quad (8.3)$$

for $\nu = 1, 2, \dots, n$, where $\omega_\nu > 0$ are the relaxation frequencies. For more details on visco-elastic modeling and the numerical method used by *WPP*, we refer to the paper by Petersson and Sjo-green [14] and the references therein.

There are three components in each of the vector variables \mathbf{u} and $\bar{\mathbf{u}}^{(\nu)}$, $\nu = 1, 2, \dots, n$, resulting in $3 + 3n$ differential equations for as many dependent variables. Hence, visco-elastic modeling will require more memory and more CPU-time, compared to the purely elastic case.

The material parameters μ_ν and λ_ν , as well as the relaxation frequencies ω_ν are determined by Emmerich and Korn's [5] least-squares procedure. In this approach, the material parameters are selected such that the quality factors Q_S and Q_P become close to constant over a frequency range

$$\omega_{min} \leq \omega \leq \omega_{max}.$$

Because the computational cost of viscoelastic modeling increases with the number of mechanisms, n , it is desirable to use the smallest value of n that gives acceptable accuracy in the approximation of $Q(\omega)$. In Figure 8.1, we present $Q(\omega)$ when the material coefficients are chosen to approximate $Q = 100$ in the frequency band $\omega \in [1, 100]$. Clearly, $n = 2$ provides inadequate modeling of a constant Q over two decades in frequency, but $n = 3$ gives a much better approximation. Increasing

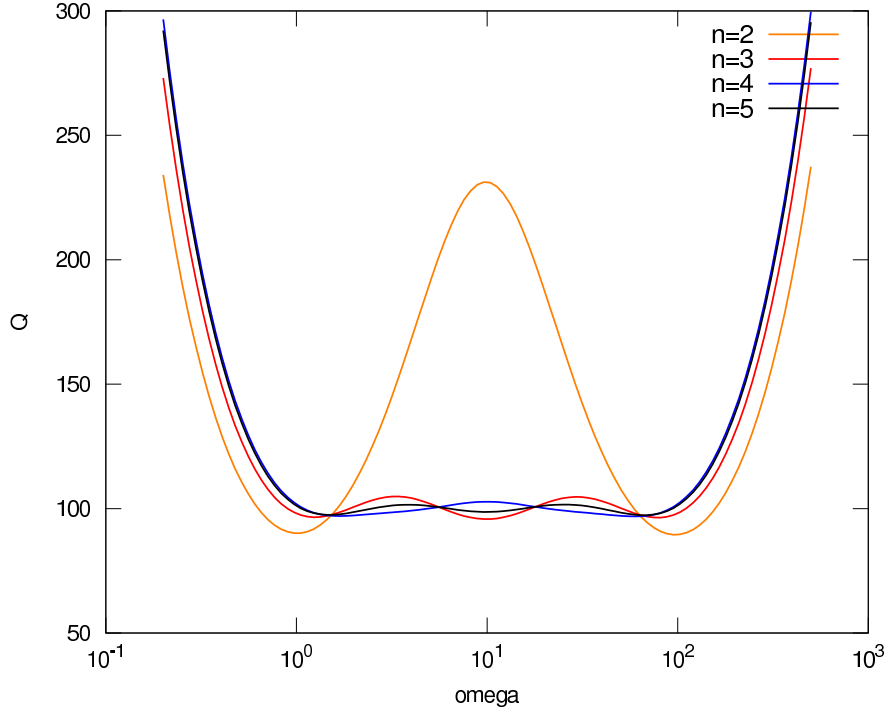


Figure 8.1: Actual quality factor $Q(\omega)$ approximating $Q_0 = 100$ in the frequency band $\tilde{\omega} \in [1, 100]$, for different numbers of viscoelastic mechanisms.

n further only leads to small improvements. It is interesting to note that in all models, $Q(\omega)$ grows rapidly for $\omega > \omega_{max}$. Hence the viscoelastic model does *not* provide significant damping of higher (poorly resolved) frequencies in the numerical solution, and does *not* act as an artificial dissipation.

Wave propagation in visco-elastic materials is dispersive, i.e., the phase velocity of a wave depends on its frequency. Figure 8.2 illustrates that the frequency dependence on the phase velocity becomes more pronounced when Q gets smaller. Also note that the phase velocity grows approximately linearly on a logarithmic scale in ω , throughout the frequency band $[\omega_{min}, \omega_{max}]$. Outside this band, the phase velocity tends to constant values. Due to the dispersive nature of visco-elastic materials, it is necessary to specify the reference frequency, ω_r , at which the phase velocities are specified.

In *WPP*, visco-elastic modeling is enabled by the `attenuation` command. For example, the command

```
attenuation nmech=3 phasefreq=2.5 maxfreq=10
```

enables visco-elastic modeling with three SLS mechanisms, tuned for the max frequency $f_{max} = 10$ Hz, such that the phase velocities are valid at the reference frequency $f_r = 2.5$ Hz. For simplicity, the lower frequency in the modeling is always two orders of magnitude smaller than the max frequency,

$$f_{min} = \frac{f_{max}}{100}.$$

Instead of using `maxfreq`, the upper frequency limit can alternatively be specified through the `minppw` option. In this case the material model is first evaluated to find $\min V_S/h$. The upper

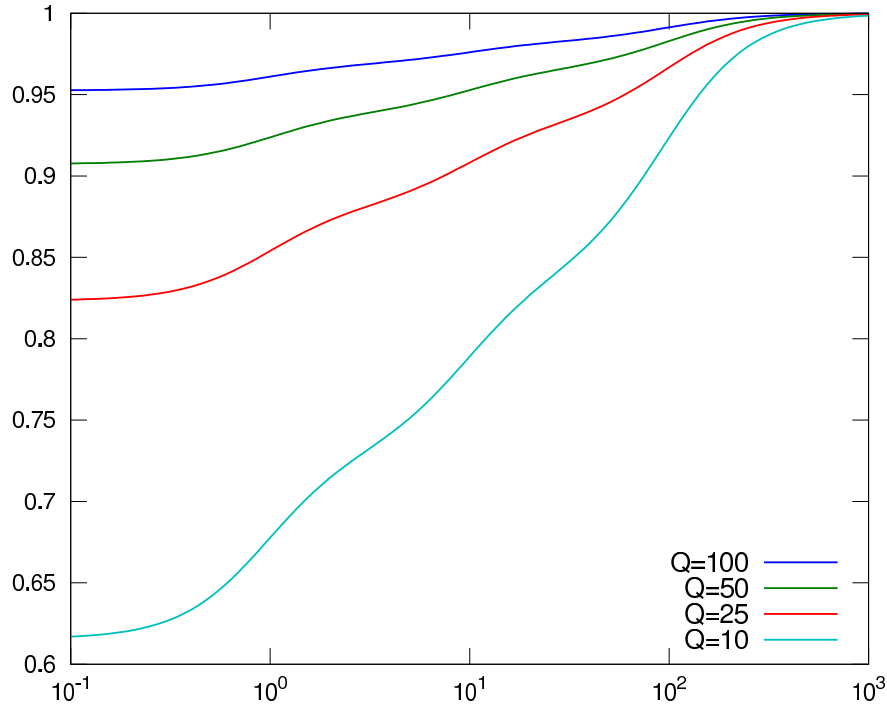


Figure 8.2: Relative phase velocity over the frequency band $\omega \in [1, 100]$. Here, $n = 3$, and the different colors correspond to different values of Q .

frequency limit is then calculated through the relation $P = \min V_s / (hf)$, i.e.,

$$f_{max} = \frac{1}{P_{min}} \min \frac{V_s}{h}, \quad f_{max} = \text{maxfreq}, \quad P_{min} = \text{minppw}.$$

The syntax for using `minppw` is given by

```
attenuation nmech=3 phasefreq=2.5 minppw=5
```

After the input file has been parsed, *WPP* outputs basic information about the attenuation modeling:

```
*** Attenuation parameters calculated for 3 mechanisms,
    max freq=2.000000e+00 [Hz], min_freq=2.000000e-02 [Hz], \
    velo_freq=1.000000e+00 [Hz]
```

Note that $\text{max_freq} = \omega_{max} / (2\pi)$, etc.

Chapter 9

Output options

9.1 Setting the output directory

The `fileio` command can be used to specify the directory where *WPP* writes its output files. If the directory does not exist, *WPP* attempts to create it for you. The `fileio` command may also be used to set the level of diagnostic messages (verbose) and how often the time step information is printed. For example,

```
fileio path=wpp_dir verbose=1 printcycle=10
```

causes all output files to be written to the directory `"/wpp_dir"`, turns on some extra diagnostic messages (a higher value gives more details), and prints the time step information every 10 time steps.

Serial and Parallel file systems Some parallel machines have a dedicated parallel file system that allows many processors to simultaneously write to the same file. These file systems are often mounted on a special directory. By default, *WPP* assumes a serial file system and will only allow one processor write to the same file at the same time. If you have access to a parallel file system, the I/O performance of *WPP* can sometimes be improved by allowing several processors to simultaneously write a file. You enable this feature by using the `pfs=1` option,

```
fileio pfs=1 path=/p/lscratcha/my_output_directory
```

Note that many parallel file systems are only accessible from dedicated directories. Enabling `pfs=1` without re-directing the output to a parallel file system can cause *WPP* to either crash or hang.

9.2 Time-history at a receiver station: the `sac` command

WPP can save the time-history of the solution at a receiver station located anywhere in the computational domain. The basic command looks like this:

```
sac x=100e3 y=50e3 z=0 file=sta1
```

Here, *WPP* saves the three components of the solution at the grid point that is closest to the specified (x, y, z) location. By default, *WPP* saves the data using the binary Seismic Analysis Code (SAC) [6] format. In the above case, the `sac` command results in three files:

```
sta1.x sta1.y sta1.z
```

The x,y,z files hold the solution time-history in the corresponding coordinate direction. Note that a positive z -component corresponds to a downward motion.

The location of the receiver station can alternatively be given in geographical (latitude, longitude, depth) coordinates. Information about the event date, time, and station name is saved in the header of a SAC file. By default the date and time are set to the date and time at the start of the simulation; the default station name is the name of the file. The default values of these fields can be changed by using the `eventData`, `eventTime`, and `sta` options,

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 \
  eventDate=2003/11/22 eventTime=16:17:00 sta=EKM
```

Note that `depth` specifies the depth of the receiver below the topography. To place a receiver at elevation e relative to mean sea level (e is negative below sea level) you use the option `z=-e`.

By default, SAC files are written to disk every 1000 time steps, and at the end of the simulation. We can change this frequency by using the `writeEvery` option. For example, to write the SAC file every 100 time steps, you would say

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 writeEvery=100
```

By default, *WPP* outputs the three components of the solution $\mathbf{u}(\mathbf{x}_r, t) = (u_x, u_y, u_z)^T$. By using the `velocity=1` option, *WPP* instead outputs the three components of the time-derivative of the solution, i.e., the velocity if *WPP* is setup to solve for displacements. The `nsew=1` option can also be used to tell *WPP* to rotate the solution components to the East, North, and vertical (positive up) directions,

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 velocity=1 nsew=1
```

The angle between North and the x -axis is determined by the azimuth (`az=...`) angle in the grid command:

```
grid x=100e3 y=50e3 z=30e3 lat=37.5 lon=-122.0 az=135
```

To remind the user of what quantities are saved in a sac file, we modify the file name extensions according to the following table:

	velocity=0	velocity=1
nsew=0	.x, .y, .z	.xv, .yv, .zv
nsew=1	.e, .n, .u	.ev, .nv, .uv

WPP can output the time history of the divergence or the curl of the solution at the receiver station, instead of the solution itself. The `variables` option governs this behavior. For example,

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 variables=div
```

will output a single file named `sta1.div` containing the divergence of the solution. Similarly

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 variables=curl
```

will output the three files `sta1.curlx`, `sta1.curly`, and `sta1.curlz`. The `velocity` and `nsew` options also work together with divergence and curl. Setting `velocity=1` will output the time derivative of any quantity selected by the `variables` option. The `nsew` option has no effect on the scalar divergence field, but `nsew=1` will make *WPP* output a representation of the curl vector in the East, North, and vertical components. The file name extensions for the div and curl variables are described in the following table:

	variables=div		variables=curl	
	velocity=0	velocity=1	velocity=0	velocity=1
nsew=0	.div	.vdiv	.curlx, .curly, .curlz	.vcurlx, .vcurlly, .vcurlz
nsew=1	.div	.vdiv	.curle, .curln, .curlu	.vcurle, .vcurln, .vcurlu

WPP can output the time history of the strain components of the solution at the receiver station. The `variables` option governs this behavior. For example,

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 variables=strains
```

will output the six files `sta1.xx`, `sta1.yy`, `sta1.zz`, `sta1.xy`, `sta1.xz`, and `sta1.yz` in sac format containing the six Cartesian components of the symmetric strain tensor. When `velocity=1`, the strains of the time derivative of the solution are output on files named `sta1.vxx`, `sta1.vyy`, etc. There is currently no support for outputting the strains in East, North, and vertical coordinates.

WPP can also output receiver time-histories in an ASCII text format,

```
sac lat=38.25 lon=-122.20 depth=0 file=sta1 sacformat=0 usgsformat=1
```

The ASCII text file holds all three components (or six when using `variables=strains`) in a single file named `sta1.txt`. When the `usgsformat=1` option is used, the file gets extension `.txt` independently of the `nsew` and `velocity` options. Instead the header inside the file is modified to reflect its contents. Note that you must give the `sacformat=0` option unless you want the solution to be output in both formats.

Notes on the sac command:

- SAC files are treated in the same way on parallel and serial file systems, because the data for each SAC file originates from one processor and is always written by that processor only.
- The binary SAC format is described in Section 12.4.
- The ASCII text format is outlined in the header of those files.
- The binary SAC files can be read by the SAC program. We also provide a matlab/octave script in `tools/readsac.m`.
- The ASCII text file format can be read by the Matlab/Octave script in `tools/readusgs.m`. The ASCII text format was slightly modified with the addition of the strain output option. The Matlab/Octave script `tools/readusgsold.m` can be used to read the older version of ASCII text files.

9.3 2-D cross-sectional data: the image command

The image command saves two-dimensional horizontal or vertical cross-sectional data at a specified time level. It can be used for visualizing the solution, making the images for a movie, or checking material properties. Each image file contains a scalar field as function of the spatial coordinates in the cross-sectional plane. The scalar field can be either a component of the solution, a derived quantity of the solution, a material property, or a grid coordinate. All in all, *WPP* can output twenty-five different types of images, see Section 11.4.2 for details.

The cross-sectional plane is specified by a Cartesian coordinate (x , y , or z). The image can be written at a specific time step or at a specified time. Images can also be output at a fixed frequency, either specified by a time step interval or a time interval.

For example, the command

```
image mode=ux y=500 file=picturefile cycle=1
```

tells *WPP* to output the x -displacement component of the solution along the vertical $y = 500$ plane. The data is written to a file named `picturefile.cycle=1.y=500.ux` after the first time step (`cycle=1`). The example

```
image mode=div x=1000 file=picturefile cycleInterval=100
```

outputs the divergence of the solution field in the yz -plane at the grid surface closest to $x = 1000$. The data is written to the files

```
picturefile.cycle=100.x=1000.div
picturefile.cycle=200.x=1000.div
...
```

With this setup, one image file is output every 100 time steps.

Note that the divergence of the solution field does not contain shear (S) waves and the rotation (curl) of the solution field does not contain compressional (P) waves. These options can therefore be used to distinguish between P- and S-waves in the solution.

The `hvelmax` and `vvelmax` modes store the maximum in time of the horizontal and vertical velocity components, respectively. As these names indicate, it is assumed that the sources in *WPP* are set up for calculating displacements. The horizontal velocity is defined as $\max(|u_t^N|, |u_t^E|)$, where u^N and u^E are the displacement components in the North and East directions, respectively. The vertical velocity is $|w_t|$, where w is the displacement component in the z -direction. For these modes, the `cycleInterval` or `timeInterval` options only determine how often the maxima are written to disk; the actual accumulation of the maxima is performed after each time step.

When *WPP* is run in parallel, the data that gets saved on an `image` file originates from all processors that are intersected by the image plane. For horizontal image planes, this means all processors. To improve the I/O performance, image data is first communicated to a number of dedicated image writing processors. By default, 8 processors write each image file to disk (or all processors if *WPP* is run on fewer than 8). This number can be changed using the `fileio` command,

```
fileio nwriters=4
```

The above command tells *WPP* to use 4 processors to write each image file. For simulations which use very large number of grid points and many processors, care must be taken to make sure that enough memory is available to buffer the image data before it is written to disk.

Notes on the image command:

- By default, single precision data is saved. Double precision data can be saved by using the `precision=double` option.
- When topography is used, an image plane along the free surface is specified by the `z=0` option.
- A `mode=topo z=0` image holds the elevation (negative z -coordinate) of the raw topography. It can only be written when topography is used.
- A `mode=grid z=0` image holds the z -coordinate (negative elevation) of the grid along the free surface, which is the actual shape of the upper surface of the computational domain.
- When topography or mesh refinement is used, vertical image planes intersect all component grids in the composite grid. In this case, cross-sectional data from all component grids are stored on the image file.
- The images files are written in a binary format, see Section 12.5 for details.
- We provide matlab/octave scripts for reading image files in the `tools` directory. The basic function is called `readimagepatch.m`. A higher level interface is provided by the `imageinfo.m` and `plotimage.m` scripts.

9.4 Generating a bird's eye view of the problem domain: the `gmt` command

The Generic Mapping Toolkit (*GMT*) [15] is a set of postscript image generation programs for geophysical applications, which can be used to make plots like Figure 9.1. In the example shown here, topography information is included as well as information on the general setup of the simulation. Note that the `gmt` command causes an ASCII text file to be outputted from *WPP*. This file contains a UNIX C-shell script with commands for the `gmt` program, holding general information about the run such as geometric coordinates of the computational domain as well as locations of sources and receivers. This information often needs to be fine-tuned to suit the needs of a particular application. To have *WPP* write a *GMT* file, you give the command

```
gmt file=bolinas.gmt
```

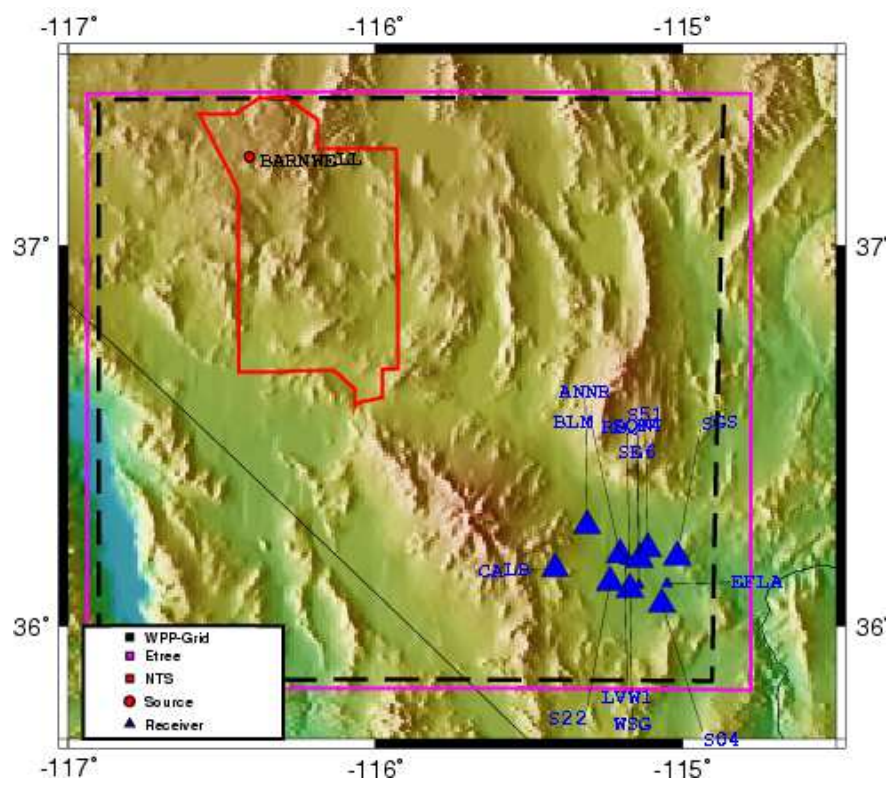


Figure 9.1: Location of the source and stations for the Barnwell simulation. This figure was generated using the GMT command, see Section 11.4.4 for details.

Chapter 10

Examples

This chapter describes most of the the input scripts in the directory `examples`. The output associated with input file `xyz.in` is given in `xyz.out`.

10.1 Lamb’s problem

The version of Lamb’s problem [9] considered here consists of a single vertical time-dependent point force acting downward on the surface of a homogeneous half-space. In this section, we use the analytical solution from Mooney [11] to test the accuracy of the numerical solution.

In the following example, the elastic half-space consists of a Poisson solid ($\lambda = \mu$) with S-wave velocity $V_s = 1000$ m/s, P-wave velocity $V_p = 1000 \cdot \sqrt{3}$ m/s, and density $\rho = 1500$ kg/m³. The elastic half-space is truncated to the computational domain

$$(x, y, z) \in [0, 8000] \times [0, 8000] \times [0, 4000].$$

The source is placed on the free surface in the center point of the horizontal plane: (4000, 4000, 0). The time dependency of the forcing is a “RickerInt” (see Figure 4.2) with $\omega = 1$ Hz, $t_0 = 2$ s and magnitude 10^{13} N. The above setup is created with the input file shown below, which can be found in `examples/lambtests/seismic1.in`

```
grid nx=161 x=8000 y=8000 z=4000
time t=5.0
fileio path=seismic1-results
block vp=1.7320508076e+03 vs=1000 rho=1500
source type=RickerInt x=4000 y=4000 z=0 fz=1e13 freq=1 t0=2
# Time history of solution
sac x=4000 y=5000 z=0 file=sta1
```

The vertical displacement at the reciever ($x = 4000$, $y = 5000$, $z = 0$) and the error for three grid sizes can be found in Figure 10.1.

The waveforms are all smooth and the problem appears to be well resolved. We present the max-norm of the errors in the vertical displacement in Table 10.1. The ratio between errors as the grid size is halved approaches 4 for the finest grid, indicating that the numerical method and the discretization of the point force are second order accurate. The center frequency in the RickerInt time function is $f_0 = 1$ Hz. Following (4.3), we estimate the highest significant frequency to be

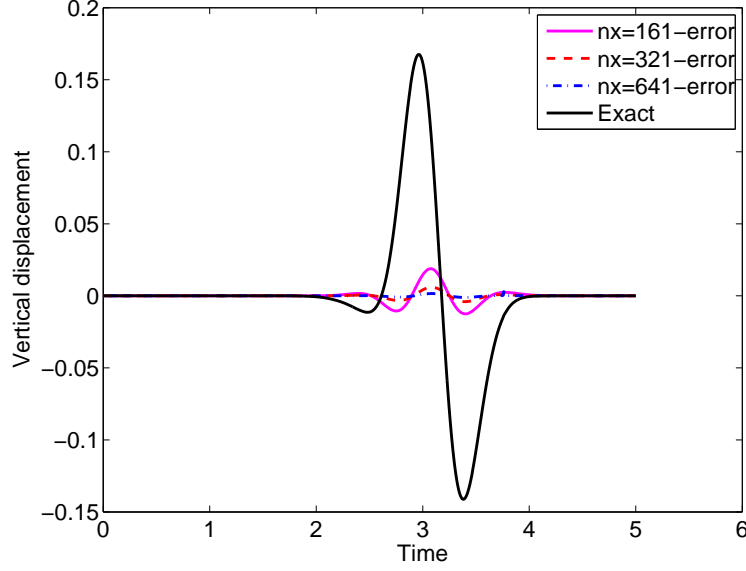


Figure 10.1: Lamb's problem: Vertical displacement at receiver $x = 4000$, $y = 5000$, $z = 0$. The black line shows the exact solution, while the magenta, red and blue lines show the errors in the numerical solutions with different grid resolution.

N_x	h	P	$\ u_z - U_z\ _\infty / \ U_z\ _\infty$	ratio
161	50	8	$1.12 \cdot 10^{-1}$	—
321	25	16	$3.54 \cdot 10^{-2}$	3.16
641	12.5	32	$9.47 \cdot 10^{-3}$	3.74

Table 10.1: Max norm errors in the vertical displacement at receiver $x = 4000$, $y = 5000$, $z = 0$.

$f_{max} = 2.5$ Hz. In this case the formula for the number of grid points per wave length (4.1) becomes $P = 1000/(2.5h)$. Note that $P = 16$ gives a relative max-norm error of about 3.5 percent.

10.2 Examples from Lifelines project 1A01: Validation of basin response codes

The following examples are taken from the Lifelines project 1A01: Validation of basin response codes, see [3]. A detailed description of the setup of the Layer over halfspace (LOH) problems can also be found in [3]. To enable a direct comparison with those results, the *WPP* simulations are set up to calculate velocities as opposed to displacements.

10.2.1 The LOH.1 problem

The LOH.1 problem, defined in the input script `examples/scec/LOH.1.h50.mr.in`, has a layered material model where the top 1000 meters ($z \in [0, 1000m]$) has different properties than the rest

of the domain. The computational domain is taken to be $(x, y, z) \in [0, 30000]^2 \times [0, 17000]$. The grid size in the base (coarsest) grid is chosen to be $h = 50$ m and the material properties in the different layers are described by

```
grid h=50 x=30000 y=30000 z=17000 extrapolate=2
block vp=4000 vs=2000 rho=2600
block vp=6000 vs=3464 rho=2700 z1=1000
```

The problem is forced by a single point moment source, positioned in the lower half-space. The time function in this problem is a Gaussian (if setup as in the input file, the Gaussian source is equivalent to using a Brune time function followed by a post processing deconvolution step, as is described in [3]). The advantage of using the Gaussian is that no post processing is necessary before comparing to the results in [3], and the Gaussian function produces less high wave number waves which are poorly resolved on the computational mesh. Note that **freq=16.6667** corresponds to the spread $\sigma = 0.06$ in the Gaussian time function. The lines to setup the source and the time duration of the simulation are:

```
time t=9
source x=15000 y=15000 z=2000 Mxy=1 m0=1e18 t0=0.36 freq=16.6667 \
      type=Gaussian
```

The layered velocity structure makes this problem an ideal candidate for mesh refinement. We align the refinement level with the material discontinuity by specifying

```
refinement zmax=1000
```

As a result the grid size in the top 1000 meters ($0 \leq z \leq 1000$) will be $h = 25$ meters. The **extrapolate=2** option in the above grid command tells *WPP* to extrapolate material properties to the ghost point and the point on the interface. This gives uniform material properties on each component grid and allows the jump conditions across the material discontinuity to be handled accurately. The **extrapolate** option should only be used when the grid interface is perfectly aligned with the material discontinuity.

The solution is recorded in an array of receivers:

```
sac x=15600 y=15800 z=0 file=sac_01
sac x=16200 y=16600 z=0 file=sac_02
sac x=16800 y=17400 z=0 file=sac_03
sac x=17400 y=18200 z=0 file=sac_04
sac x=18000 y=19000 z=0 file=sac_05
sac x=18600 y=19800 z=0 file=sac_06
sac x=19200 y=20600 z=0 file=sac_07
sac x=19800 y=21400 z=0 file=sac_08
sac x=20400 y=22200 z=0 file=sac_09
sac x=21000 y=23000 z=0 file=sac_10
```

Numerical velocity time histories for station 10 are shown in Figure 10.2 together with a semi-analytical solution. We conclude that most features in the solution are very well captured on this grid. As is customary in seismology, the velocity components have been rotated to polar components, with the origin at the source. The **sac** command outputs the u_x , u_y and u_z -components

of the velocity. These components are rotated to radial and transverse components using the transformation,

$$u_{rad} = 0.6u_x + 0.8u_y, \quad u_{tran} = -0.8u_x + 0.6u_y.$$

The vertical component is given by u_z (positive downwards).

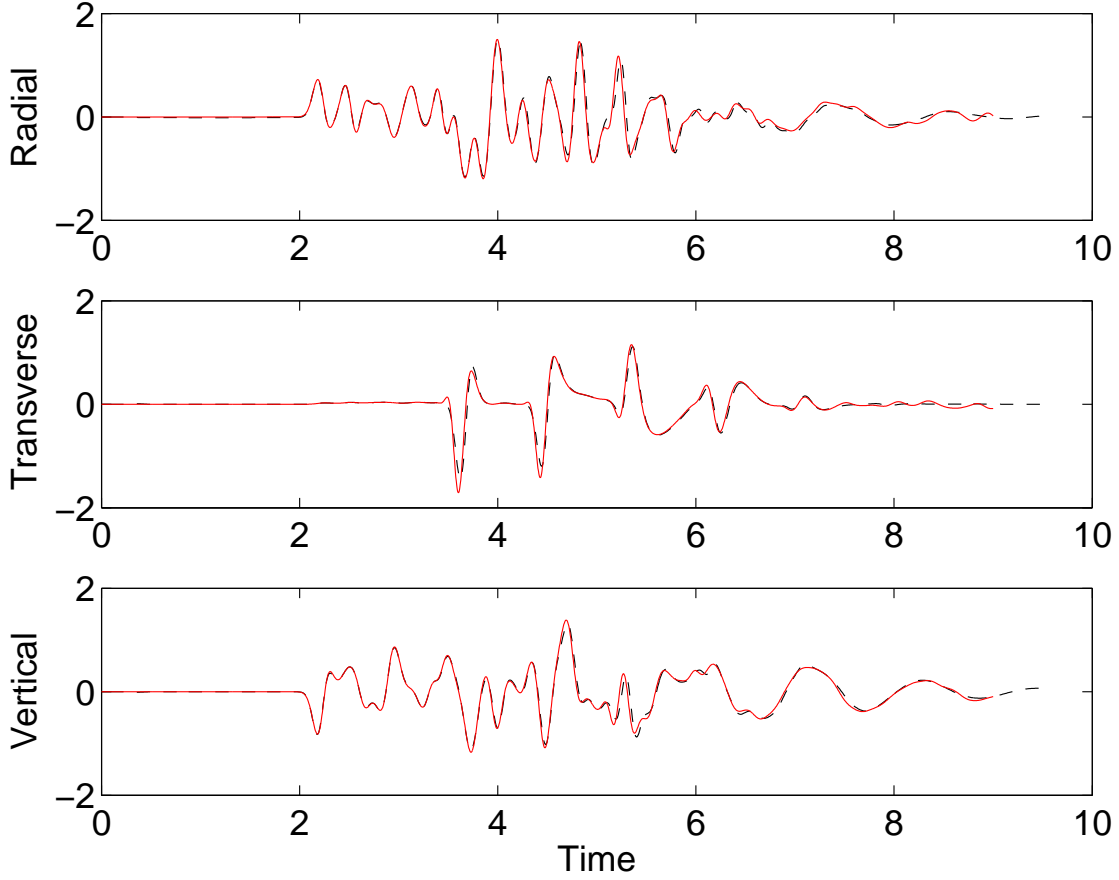


Figure 10.2: LOH.1: The radial (top), transverse (middle) and vertical (bottom) velocities for receiver number 10. Here the numerical solution is plotted in red while the semi-analytical solution is represented by dashed black lines.

By using formulas (4.1)-(4.3), we can calculate the number of points per wave length for this simulation. Since we are using a Gaussian time-function, the center frequency is $f_0 = 1/(2\pi\sigma) \approx 2.6526$ and we estimate the upper power frequency to be $f_{max} \approx 2.5f_0 = 6.6315$ Hz. The material model has $\min V_s = 2000$ m/s where the grid size is $h = 25$ m, and we arrive at

$$P = \frac{2000}{25 \cdot 6.6315} \approx 12.1.$$

From our discussion in Section 4.3, 12.1 points per wave length is on the low side, but visual inspection of Figure 10.2 indicates very good agreement of the wave forms.

10.2.2 The LOH.2 problem

The geometrical setup of the LOH.2 problem is identical to that of LOH.1, but the LOH.2 problem models an earthquake along a fault plane. The slip along the fault is modeled by a large number of point moment tensor sources with the time dependency given by the Gaussian function with different offsets in time (depending on distance from the hypocenter). The fault plane coincides with the y-z-plane in the computational grid. The input files for LOH.2 can be found in `examples/sccec/LOH.2.h50.mr.in`.

As for LOH.1, a semi-analytical solution is available for $\sigma = 0.06$ corresponding to the frequency parameter `freq=16.6667` in all source commands. This leads to the same number of grid points per wave length as for LOH.1.

$$P \approx 12.1.$$

In Figure 10.3, we evaluate the error in the solution at station 10, by comparing velocity time-histories in the numerical solution to a semi-analytical solution. We conclude that most features in the solution indeed are captured on this grid, in particular before time $t \approx 5.5$. At later times, artificial effects of the outflow boundary dominate the solution error. These effects are larger than in LOH.1 because the sources are distributed in space so some sources are closer to the outflow boundary than in LOH.1.

10.2.3 The LOH.3 problem

The LOH.3 problem, which adds effects of anelastic attenuation to the LOH.1 problem, is defined in the input script `examples/sccec/LOH.3-n3-h50.mr.in`. To reduce any artificial effects from the far-field boundaries, we use a slightly larger computational domain compared to LOH.1,

```
grid h=50 x=40000 y=40000 z=20000 extrapolate=2
```

As before, the `extrapolate=2` option means that the material properties are extrapolated to the ghost points and the point on the interface in each component grid. This option is useful for increasing the accuracy when a material discontinuity is perfectly aligned with a grid refinement boundary.

The visco-elastic modeling is enabled with the command

```
attenuation phasefreq=2.5 nmech=3 maxfreq=15
```

In this case, three standard linear solid mechanisms are used (`nmech=3`) to give a material with approximately constant quality factors in the frequency band $0.15 \leq f \leq 15$ Hz. Note that only the upper frequency limit needs to be specified (`maxfreq=15`); the lower limit is always 100 times smaller (and can not be specified). Since the visco-elastic material is dispersive, we use the `phasefreq=2.5` option to specify at what frequency the compressional and shear speeds should apply. In the description of the LOH.3 problem, see Day et al. [4], this frequency is given as 2.5 Hz.

Apart from the density and the material velocities, the material model must include the quality factors for the attenuation of compressional (Q_P) and shear waves (Q_S). For this problem, we use the block commands

```
block vs=3464 vp=6000 r=2700 Qs=69.3 Qp=155.9
block vs=2000 vp=4000 r=2600 z2=1000 Qs=40 Qp=120
```

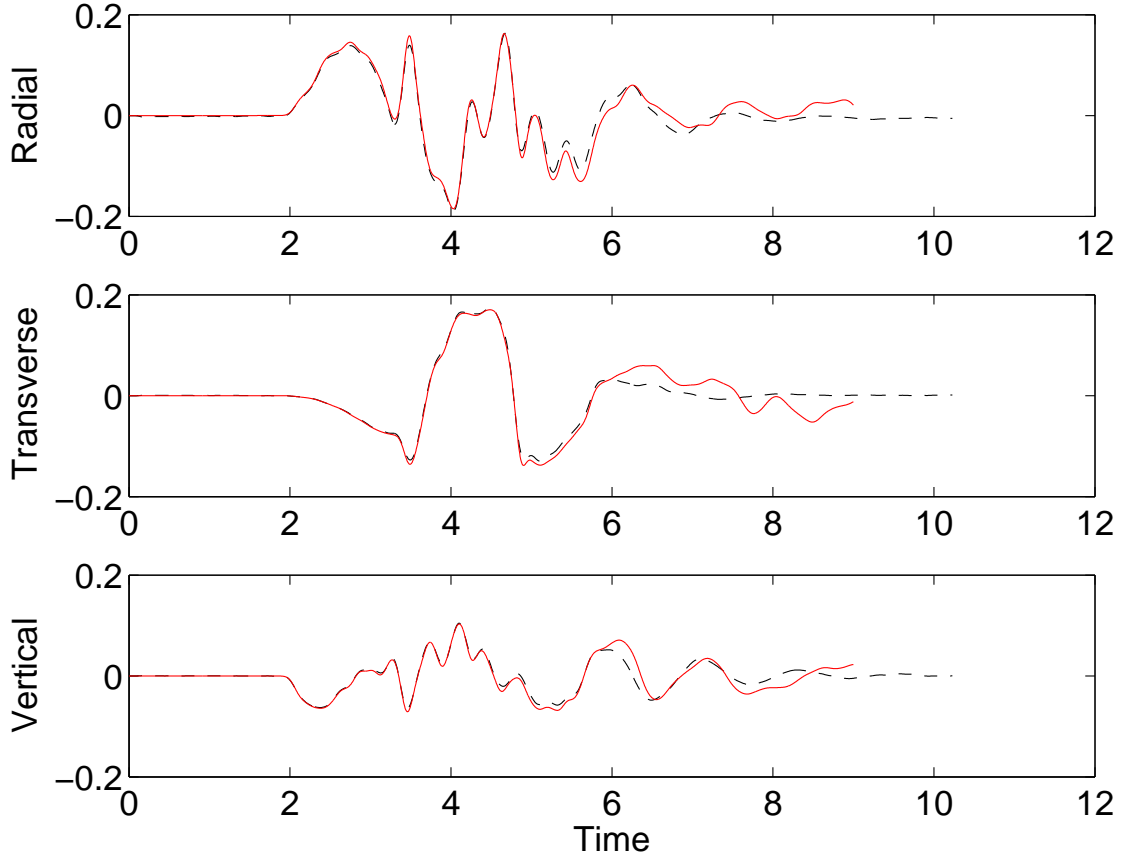


Figure 10.3: LOH.2: The radial (top), transverse (middle) and vertical (bottom) velocity components recorded at station number 10. Here the dashed black line is a semi-analytical solution and the red line is the numerical solution.

Similar to the LOH.1 test case, the source is of point moment-tensor type with a Gaussian time-function. Note, however, that for LOH.3, the Gaussian has spread $\sigma = 0.05$, corresponding to the center angular frequency $\omega_0 = 1/\sigma = 20$ and center frequency $f_0 = 20/(2\pi) \approx 3.18$ Hz. The source is specified by the command

```
source x=20000 y=20000 z=2000 Mxy=1 m0=1e18 t0=0.2 freq=20 \
type=Gaussian
```

Similar to the LOH.1 problem, the layered velocity structure makes the LOH.3 problem an ideal candidate for mesh refinement, and we use the same **refinement** command as before. The grid size is therefore $h = 25$ meters in the top 1000 meters of the model, and $h = 50$ meters elsewhere.

The solution is recorded in the same array of receivers as before, but all (x, y) coordinates have been incremented by 5000 meters to be in the same position relative to the source. Results recorded at station 10 are shown in Figure 10.4. As for the LOH.1 problem, the velocity is presented in polar components. We conclude that the numerical solution is in very good agreement with the semi-analytical solution shown in black.

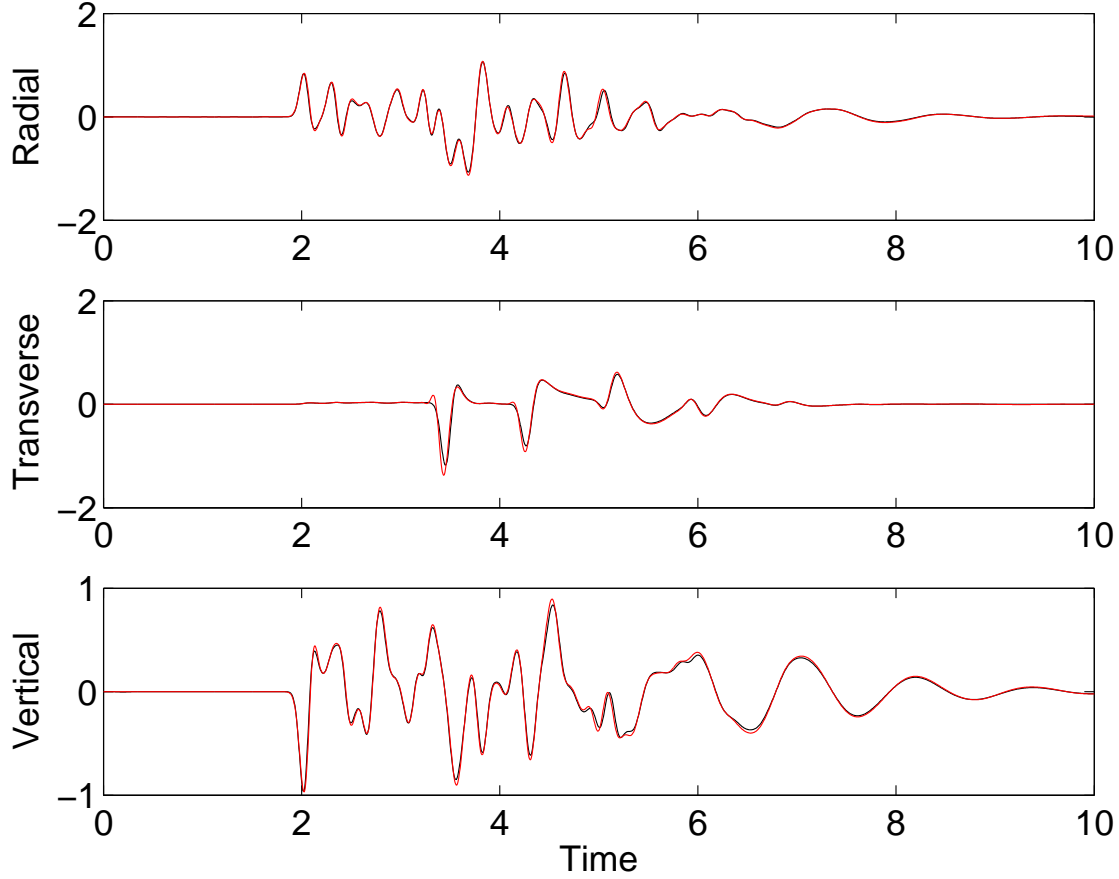


Figure 10.4: LOH.3: The radial (top), transverse (middle) and vertical (bottom) velocities at receiver number 10. Here the numerical and semi-analytical solutions are plotted in red and black, respectively.

As before we can estimate the resolution in terms of the number of grid points per shortest significant wave length. In this case the center frequency is $f_0 \approx 3.18$ Hz and we estimate the upper power frequency to be $f_{max} \approx 2.5f_0 = 7.95$ Hz. The material model has $\min V_s = 2000$ m/s where the grid size is $h = 25$ m, and we arrive at

$$P = \frac{2000}{25 \cdot 7.95} \approx 10.06.$$

10.3 The Grenoble basin test case

This example uses realistic topography and mesh refinement to model a scenario earthquake near Grenoble, France. The *WPP* input file for this simulation is called **Grenoble.in** and can be found in the **examples/ifile** directory.

Grenoble is located in a Y-shaped valley in the foothills of the Alps. The extent and geographical orientation of the computational domain is described by

```
grid x=40e3 y=43e3 z=40e3 lon=5.52 lat=45.01 az=0 h=200
```

Hence, the x -axis points in the direction of North ($az=0$) and the y -axis is directed due East. With this orientation, the `lon` and `lat` options specify the location of the South-West corner of the computational domain.

To setup the topography we give the command

```
topography input=grid file=grenobleCoarse.topo zmax=3000 order=2
```

The file `grenobleCoarse.topo` holds the elevation (in meters) above mean sea level on a regular grid in geographical (lat-lon) coordinates. A plot of the topography can be found in Figure 6.2. The material properties are described by a heterogeneous model with granite and sediment. The properties of the granite are assumed to only depend on depth, and are setup using block commands,

```
block vp=5600 vs=3200 rho=2720
block vp=5920 vs=3430 rho=2720 z1=3e3
block vp=6600 vs=3810 rho=2920 z1=27e3
block vp=8000 vs=4450 rho=3320 z1=35e3
```

Since the material commands are read in the order they occur, the properties of the top 3000 meters are described by the first `block` command, and the subsequent `block` commands describe the properties deeper into the earth because their extent is restricted by the `z1` option (note that `z1` and `z2` correspond to depth in the presence of topography).

The geometry of the sedimentary basin is described by the `ifile` command,

```
ifile filename=bedrock_surface.dat
```

In this case, the file `bedrock_surface.dat` holds the depth of the sedimentary basin on a regular grid in geographical (lon-lat) coordinates. Note that this grid is unrelated to the computational grid and the grid used in the topography file. The format of this ASCII text file is described in Section 12.3. The `ifile` command can be used to describe the depths of several material surfaces, but in this case we only have one.

For each material surface in the `ifile` command, there must be a `material` command with a unique `id` number. The material properties between the free surface and the first material surface in the `ifile` are defined by the `material` command with the lowest `id` number, and so on. The `ifile` command only assigns material properties down to the depth of the last material surface.

In our case, the material properties of the sediment, i.e., between the free surface and the single material surface in the `ifile` command are described by

```
material id=1 vs=300 vp=1450 vpgrad=1.2 rho=2140 rhograd=0.125 \
      qs=50 qp=876
```

Note that the depth of the material surface is relative to the topography, which means that no sediment is present where the depth is zero. By plotting the compressional wave speed along the top surface and in a vertical cross-section, we can see both the horizontal extent and the variable depth of the sedimentary basin, see Figure 10.5.

The slowest shear speed in the model is 300 m/s (in the sediment) and the fastest compressional speed is 8,000 m/s (in the granite deeper than 35 km). To reduce the ratio between the shortest and longest wave in the solution, we can impose a minimum threshold on the material velocities through the command

```
globalmaterial vpmin=800 vsmin=500
```

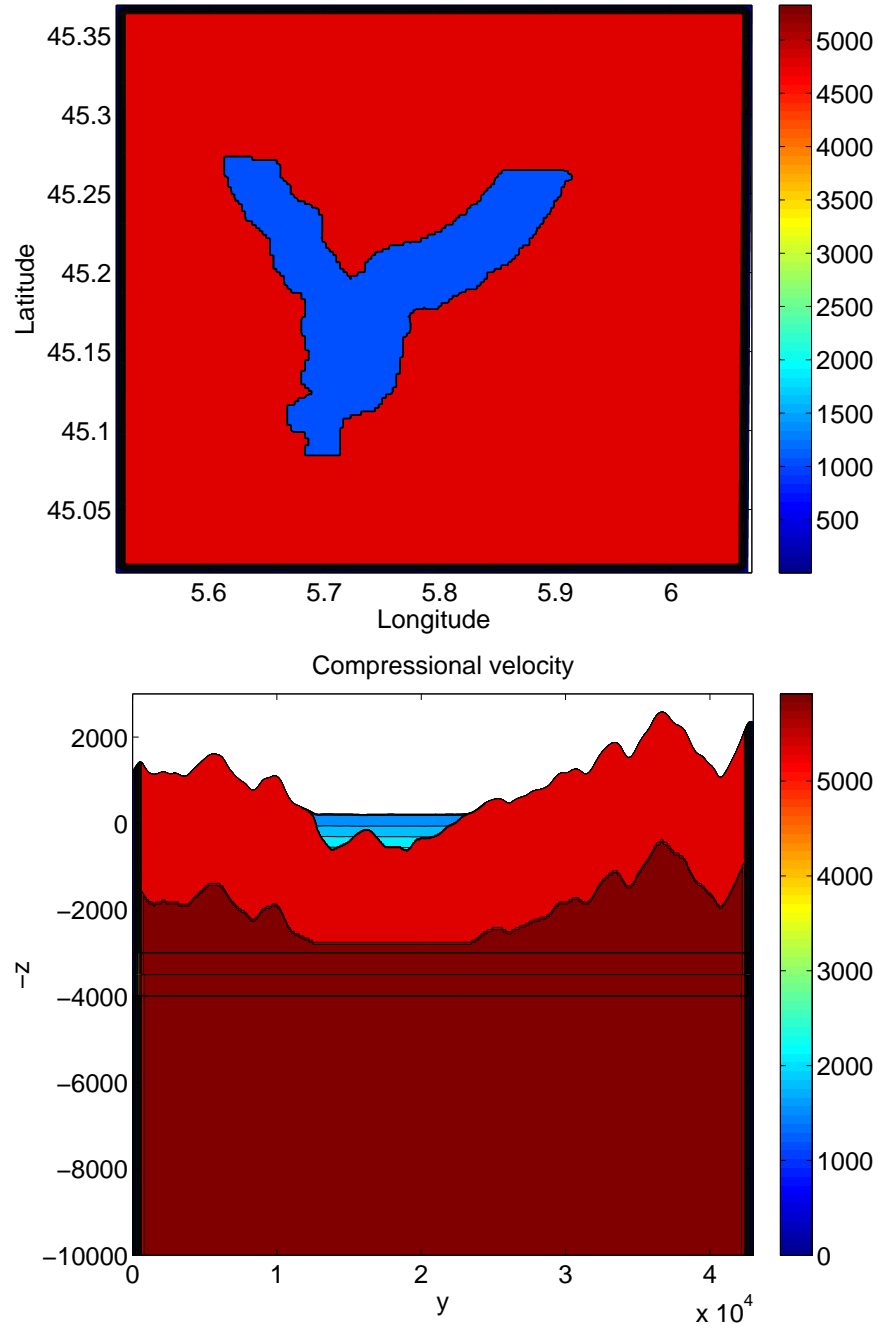


Figure 10.5: Grenoble basin model. Top: compressional velocity along the free surface, illustrating the extent of the sedimentary basin. Bottom: compressional velocity in the vertical cross-section $x = 20,000$ (latitude $\approx 45.2^\circ$). The black horizontal lines mark the bottom of the curvilinear grid and the mesh refinement interfaces for the Cartesian component grids.

The slow material in the sedimentary basin sets the grid size requirement for the simulation. To reduce the total number of grid points, we use local mesh refinement to coarsen out the computational grid below the curvilinear grid (that extends to `zmax= 3000`),

```
refinement zmax=3500
refinement zmax=4000
```

Since the base grid has grid size $h = 200$ m, the two refined Cartesian grids get grid sizes $h = 100$ m and $h = 50$ m, respectively. The average grid size in the curvilinear grid, which covers the sedimentary basin, is therefore $h = 50$ m.

The source mechanism in this small ($M_w = 2.9$) scenario earthquake corresponds to case W1 in the ESG2006 benchmark,

```
source lat=45.2167 lon=5.9167 topodepth=3e3 m0=2.8184e13 \
      strike=45 dip=90 rake=180 type=GaussianInt \
      freq=188.56 t0=0.06
```

Note the very high angular frequency in the GaussianInt time function, corresponding to a center frequency of $f_0 \approx 30.0$ Hz. The corresponding upper power frequency would be $f_{max} \approx 75$ Hz. With a minimum shear speed threshold of 500 m/s, this corresponds to a smallest wave length of $L_{min} = 500/75 \approx 6.67$ m, making a fully resolved simulation extremely challenging. Instead of artificially lowering the `freq` parameter, we choose to filter the source time function with a two-pass two-pole Butterworth filter with corner frequency 1 Hz,

```
prefilter fc=1.0
```

The resulting motion is identical to running the simulation with the original time function followed by time filtering the motion at all points in space using the same filter.

If we use this corner frequency to estimate the shortest wave length, we get $L_{min} \approx 500$ m, and the curvilinear grid size $h = 50$ should provide acceptable resolution with $P = 10$ grid points per wave length.

In order to avoid incompatibilities due to the exponentially decaying tails in the filtered source time function, *WPP* automatically adjusts the starting time of the simulation from $t_0 = 0$ to $t_0 \approx -3.97$. As a result the requested 15 seconds of simulation time corresponds to final time $t_0 + 15 \approx 11.03$. The peak horizontal velocities along the free surface are shown in Figure 10.6, and the calculated velocity time history at station R06 is shown in 10.7.

The input file `Grenoble.in` is setup to report the motion at many other stations, and to save several different image files of the material model and the solution. We encourage the reader to run this case and to further explore the results.

10.4 Modeling the October 2007, Alum Rock earthquake

This example uses both realistic topography and mesh refinement. The material properties and topography are obtained from an Etree data base developed by the USGS. Hence, before you can run this case, you must download the data base from the USGS website, see Section 5.2. It is also necessary to configure *WPP* to use the `efile` command, see Section A.5.

The input scripts are located in the directory `wpp-version-2.2/examples/efile`:

```
Alumrock.in Alumrock-2.in Alumrock-2q.in
```

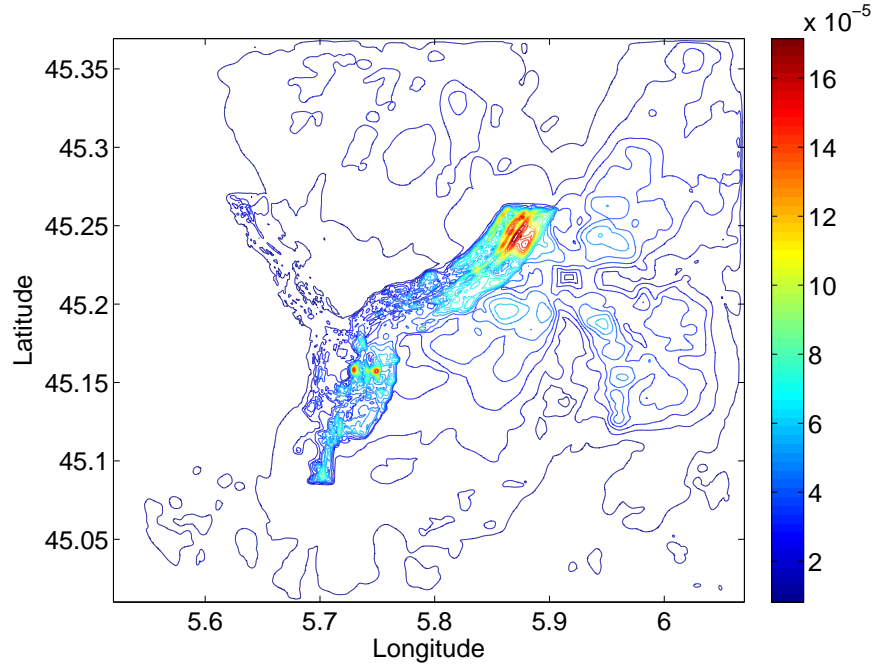


Figure 10.6: Grenoble scenario $M_w = 2.9$ earthquake. Peak horizontal velocity [m/s] for lowpass filtered motion with corner frequency $f_c = 1.0$ Hz. Note the elevated velocity levels in the sedimentary basin.

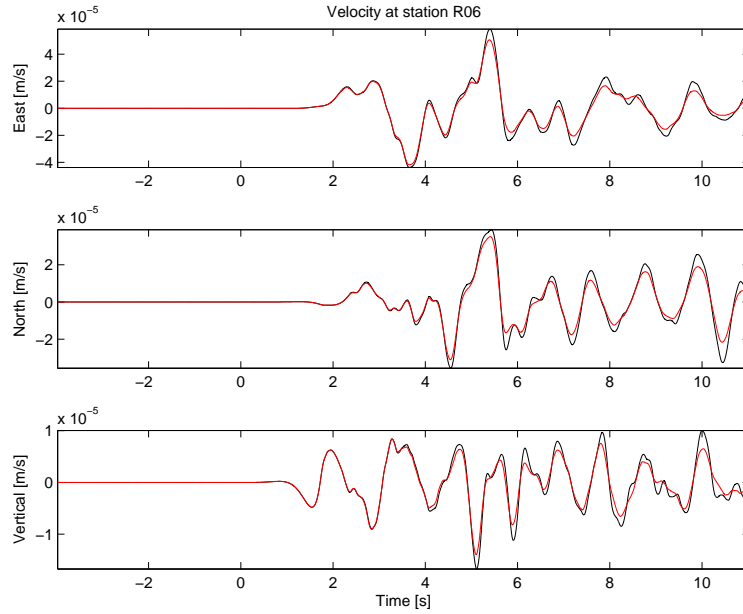


Figure 10.7: Grenoble scenario $M_w = 2.9$ earthquake. Velocity time history at receiver station R06 located at (lon=5.8210, lat=45.2086, depth=0). The black and red lines correspond to the purely elastic and the visco-elastic material model, respectively.

The main difference between the two first files is the grid size. The case in the `Alumrock.in` file uses only about 729,000 grid points and can easily be run on a workstation. The grid size in the top grid is $h = 500$ m, so this simulation can only be expected to capture very long period motions (frequencies up to 0.1 Hz). The `Alumrock-2.in` case uses half the grid size, leading to about 5.4 Million grid points and captures frequencies up to 0.2 Hz. This case can also be run on a workstation as long as it has enough memory, but will take about 16 times longer to execute once the material properties have been read from the Etree. Note that the Etree can take a long time to read, so be patient while the material model is being setup.

The source model for this magnitude $M_w \approx 5.4$ earthquake is discretized by many moment tensor sources distributed over the fault plane with variable strength and initiation times. Similar to previous examples, the input files are setup to save several image and sac files. As an example, in Figures 10.8-10.9, we show peak horizontal velocities along the top surface as well as the time history of the motion. The reader is encouraged to run this case to further explore the results. If you have access to a larger parallel machine, you can easily capture higher frequencies in the motion by reducing the grid size in the grid command and increasing the corner frequency in the prefilter command.

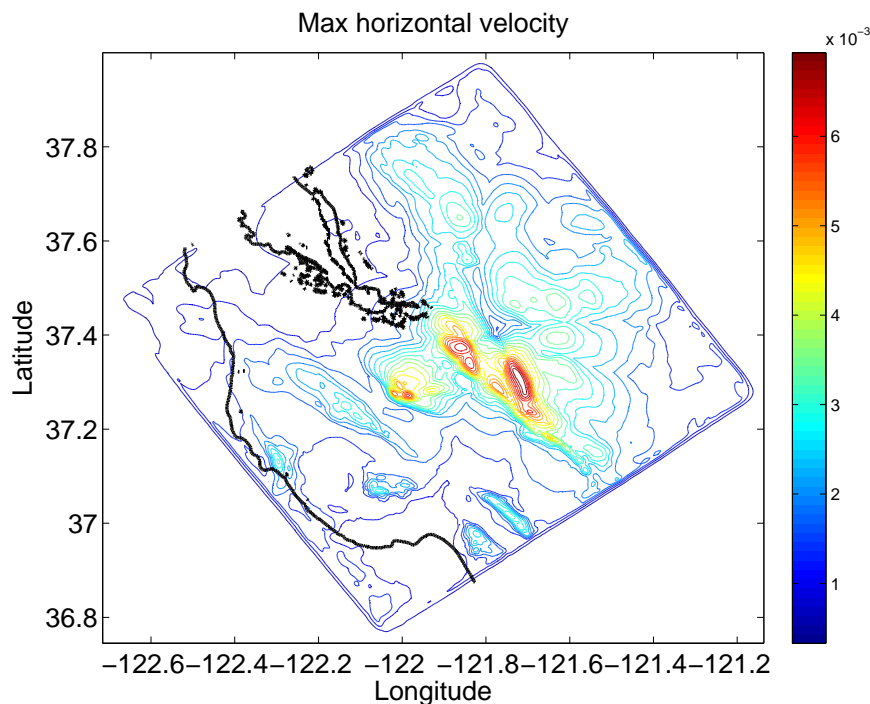


Figure 10.8: Alum Rock $M_w \approx 5.4$ earthquake. Max horizontal velocity for lowpass filtered motion with corner frequency $f_c = 0.2$ Hz. The coast line of southern San Francisco bay and the Pacific ocean is outlined with a thicker black line.

The third input file, `Alumrock-2q.in`, uses a visco-elastic material model, but is otherwise the same as `Alumrock-2.in`. The ground motion time history at one recording station is shown in Figure 10.9. At this low frequency, the main influence of the visco-elastic material model is a reduction of the amplitudes at later times. Also note that the time histories start at time ≈ -20

instead of time zero. This is side effect of using the `prefilter` command.

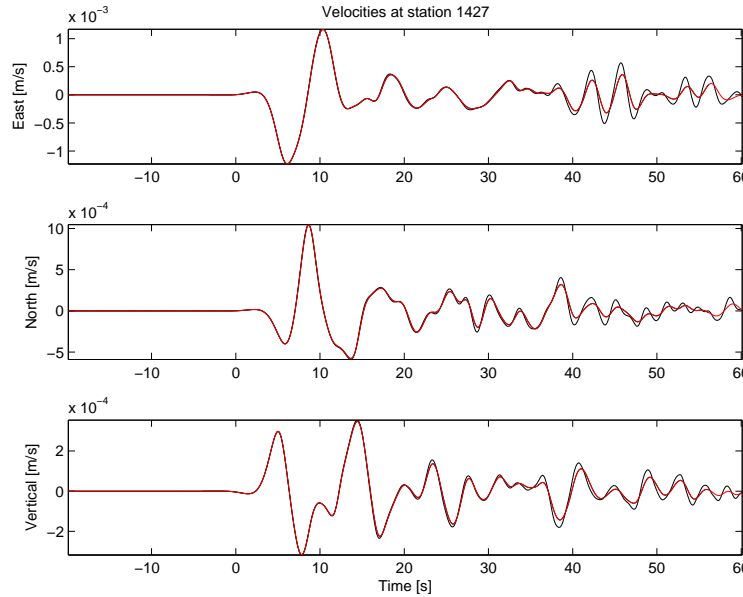


Figure 10.9: Alum Rock $M_w \approx 5.4$ earthquake. Simulated velocity time history at station 1427 (Longitude 122.025 W, Latitude 37.402 N, zero depth). The black and red lines show the purely elastic and visco-elastic responses, respectively.

10.5 A scenario earthquake in the Caucasus region

This example demonstrates the use of a pfile material model together with a topography grid file. The input and output files can be found in

`examples/pfile/caucasus.in` `examples/pfile/caucasus.out`

The pfile material model is saved according to the file format in Section 12.2. Here we set up the computational domain and read the file `caucasus_mod3.ppmod` using the commands

```
grid x=160e3 y=160e3 z=50e3 lon=47.08 lat=38.04 az=0 h=500
pfile filename=caucasus_mod3.ppmod
```

Note that the origin and extent of the grid must be covered by the pfile region, unless other material commands are used. By default, `smoothing_size=5` is used in the `pfile` command, which works well when a coarser material model is combined with a much finer grid resolution. For example, the shear wave speed along the free surface is shown on the left side in Figure 10.10. While reading the pfile, *WPP* outputs some general information about the content on the file. In this case, we get

```
*** Reading data from Pfile caucasus_mod3.ppmod in directory ./
Pfile model name (string): 'Caucasus'
Step size in lat and lon: 0.25
```

```

Number of latitude points: 7
Min Lat: 38 Max Lat: 39.5
Number of longitude points: 19
Min Lon: 44.5 Max Lon: 49
Number of depth points: 30
Min depth: 0 Max depth: 161
Optional indices: Sediment: -99 MoHo: -99 410: -99 660: -99
Attenuation Q-factors available: yes

```

We define the topography by

```
topography input=grid file=caucasus.topo zmax=7560 order=2
```

After reading the topography grid file, *WPP* outputs some general information,

```

***inside extractTopographyFromGridFile***
Nlon=108 Nlat=90
lonMin=4.204170e+01, lonMax=5.095830e+01
latMin=3.704170e+01, latMax=4.445830e+01
elevMin=0.000000e+00, evalMax=3.755000e+03

***Topography grid: min z = -3.681038e+03, max z = 6.142771e+01, \
    top Cartesian z = 7.560000e+03

```

It is important that the geographical location of the topography grid file is consistent with the **grid** command such that the elevation can be defined for all grid points on the top of the computational domain. Note that the raw topographic information has elevation '*e*' (positive above mean sea level) in the range $0 \leq e \leq 3755$. The topography gets smoothed before the computational grid is constructed and, as a result, the *z*-coordinate (negative above mean sea level) satisfies $-3681.038 = \tau_{min} \leq z \leq \tau_{max} = 61.42771$. These bounds are slightly different from the bounds on the elevation. As before, we must choose the thickness of the curvilinear grid such that the **zmax** option in the **grid** command satisfies $\mathbf{zmax} \geq \tau_{max} + 2(\tau_{max} - \tau_{min})$. The topography is shown on the right side of Figure 10.10.

Once the computational grid has been constructed, *WPP* outputs general information about the number of grid points in each component,

```

Global grid sizes (without ghost points)
Grid      h      Nx      Ny      Nz      Points
  0       500    321     321     83     8552403
  1       250    641     641      7     2876167
  2       250    641     641     38     15613478
Total number of grid points (without ghost points): 2.7042e+07

```

Even though we ran this calculation on 32 processors, it is probably possible to fit it in memory on a larger workstation, at least for a purely elastic calculation.

We follow the general velocity structure and coarsen out the grid below depth $z \geq 9000$,

```
refinement zmax=9000
```

After populating all grid points with material properties, *WPP* outputs the ranges,

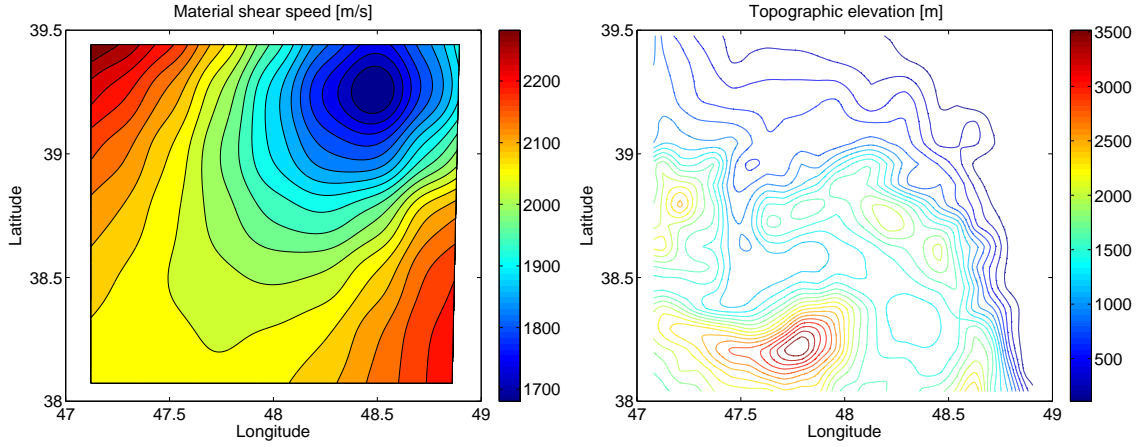


Figure 10.10: Shear wave speed (left) and topographic elevation (right) along the surface for the Caucasus region.

```

----- Material properties ranges -----
2457.72 kg/m^3 <= Density <= 3012.49 kg/m^3
3105.35 m/s    <= Vp      <= 7762.5 m/s
1680.37 m/s    <= Vs      <= 4195.49 m/s
1.84513        <= Vp/Vs   <= 1.85425
6.93972e+09 Pa <= mu     <= 5.30257e+10 Pa
9.82084e+09 Pa <= lambda  <= 7.5468e+10 Pa
-----

```

Furthermore, *WPP* outputs resolution information that can be translated into the number of grid points per shortest wave length,

```

***** PPW = minVs/h/maxFrequency *****
g=0, h=5.000000e+02, minVs/h=5.21109 (Cartesian)
g=1, h=2.500000e+02, minVs/h=9.56454 (Cartesian)
g=2, h=2.500000e+02, minVs/h=6.72147 (curvilinear)

```

A simple point moment tensor source is used to model a scenario magnitude $M_W = 6.6$ event, source $x=80e3$ $y=80e3$ $z=10e3$ $m_0=1e19$ $strike=45$ $dip=90$ $rake=180$ \
 type=GaussianInt freq=1.0 t0=6

Since the time dependence is a **GaussianInt** function, the dependent variables represent displacements. We estimate the fundamental frequency in the time function through equation (4.2), giving $f_0 \approx 0.16$ Hz. As a result, the upper power (highest significant) frequency satisfies $f_{max} \approx 0.4$ Hz. By using the above grid information on $\min V_S/h$, we can calculate the number of grid points per shortest wave length,

$$P = \frac{\min V_s}{h f_{max}} \geq \frac{5.21}{0.4} \approx 13.0.$$

While running the simulation we ask *WPP* to accumulate the peak horizontal velocity along the free surface, and output the result at time $t = 50$ seconds,

```
image mode=hvelmax z=0.0 time=50 file=g
```

In order to plot the horizontal velocity as function of geographical coordinates, we also output latitude and longitude image files,

```
image mode=lat z=0 cycle=0 file=g
image mode=lon z=0 cycle=0 file=g
```

After the simulation is completed, we can then use the Matlab/Octave scripts in the `tools` directory to read the image files (for example using the `readimagepatch` script). A contour plot of the max horizontal velocity is shown in Figure 10.11.

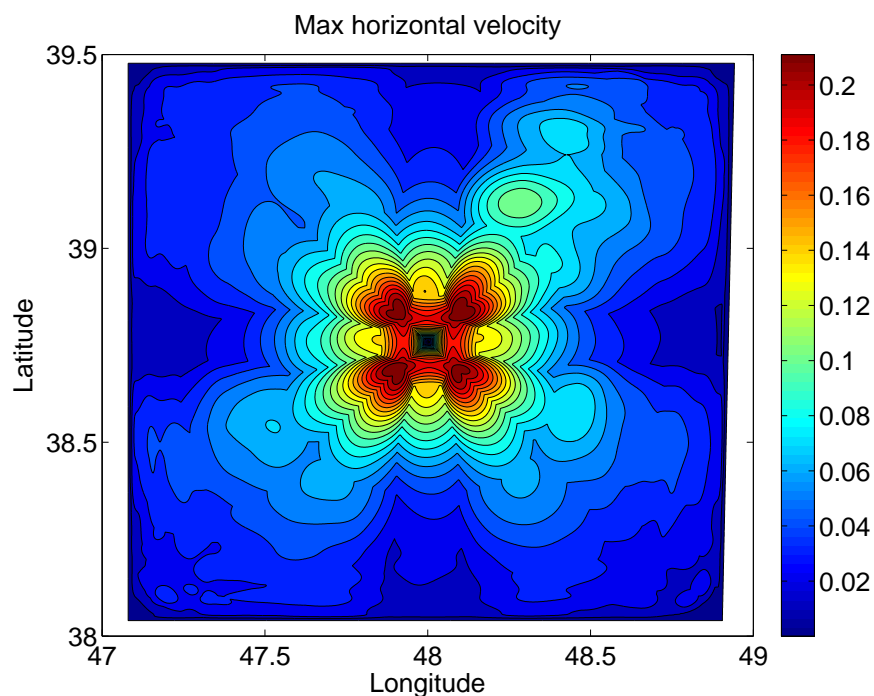


Figure 10.11: Max horizontal velocity for the Caucasus scenario earthquake

We save the velocity time history at Latitude 38.25 N, Longitude 48.295 E, using the command

```
sac lat=38.25 lon=48.295 depth=0 nsew=1 velocity=1 file=ardabil \
  usgsformat=1 sacformat=0
```

The information is saved in the USGS text format, resulting in one ASCII text file `ardabil.txt`, which can be read by `tools/readusgs.m`. The `nsew=1` option makes *WPP* save the components in the East, North, and Vertical directions, and `velocity=1` gives the velocity (as opposed to the displacement) time history. The time history is shown in Figure 10.12.

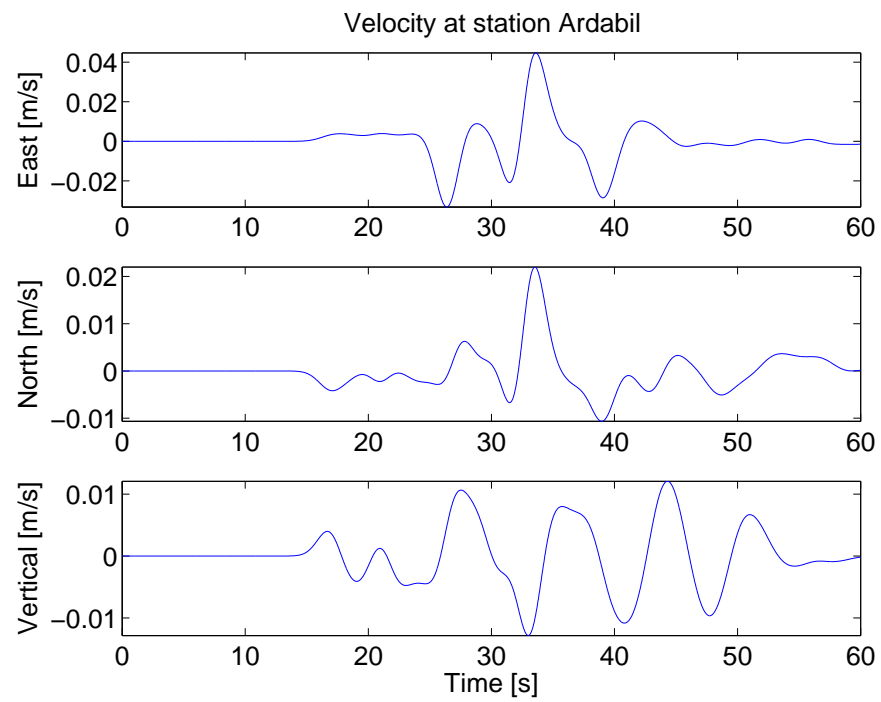


Figure 10.12: Velocity time history at Latitude 38.25 N, Longitude 48.295 E.

Chapter 11

Keywords in the input file

The syntax of the input file is

```
command1 parameter1=value1 parameter2=value2 ... parameterN=valueN
# comments are disregarded
command2 parameter1=value1 parameter2=value2 ... parameterN=valueN
...
```

Each command starts at the beginning of the line and ends at the end of the same line. Blank and comment lines are disregarded. A comment is a line starting with a `#` character. The order of the parameters within each command is arbitrary. The material commands (block, ifile, pfile, and efile) are applied in the order they appear, but the ordering of all other commands is inconsequential. Also note that the entire input file is read before the simulation starts.

Parameter values are either integers (-2,0,5,...), floating point numbers (20.5, -0.05, 3.4e4), or strings (wpp, earthquake, my-favorite-simulation). Note that there must be no spaces around the `=` signs and strings are given without quotation marks and must not contain spaces. Depending on the specific command, some parameter values are required to fall within specified ranges.

A brief description of all commands is given in the following sections. The commands marked as [required] must be present in all WPP input files, while those marked as [optional] are just that. Other commands, such as those specifying the material model can be given by (a combination of) different commands (block, pfile, efile, or ifile). Unless *WPP* is run in one of its test modes, the material must be specified by at least one of these commands and at least one source must be specified.

11.1 Basic commands

11.1.1 fileio [optional]

The **fileio** command is used for specifying an output directory, setting the amount of information outputted by *WPP*, the output frequency during the time-stepping, as well as enabling fast I/O for parallel file system. See § 9.1 for more information.

Syntax:

```
fileio path=... verbose=... printcycle=... pfs=... nwriters=...
```

Required parameters:

None

fileio command parameters			
Option	Description	Type	Default
path	path to a directory where all output will be written	string	.
verbose	sets the level of diagnostic messages written to standard out	int	0
printcycle	sets the interval for printing the cycle, time, dt info	int	100
pfs	assume a parallel (1) or serial (0) file system when writing image or volimage files (several processes can simultaneously write the same file on a parallel file system)	int	0
nwriters	set the number of processes that write an image or volimage file	int	8

11.1.2 grid [required]

Syntax:

grid nx=... ny=... nz=... x=... y=... z=... h=... lat=... lon=... az=...

Required parameters:

See below.

The grid command specifies the extent of the computational domain and the grid size in the base grid. When grid refinement is used, the base grid is the coarsest grid. Optionally the grid command also specifies the latitude and longitude of the origin and the azimuth angle between North and the x -axis.

There are three basic ways of specifying the extent of the computational domain and the grid size:

- number of grid points in all three dimensions and the grid size: **nx=... ny=... nz=... h=...**
- spatial extents in all three dimensions and the grid size: **x=... y=... z=... h=...**
- spatial extents in all three dimensions and the number of grid points in one direction (the x -direction in this example): **x=... y=... z=... nx=...**

It is not allowed to over specify the grid size. For example, if **x=...** is given, you can not specify both **h=...** and **nx=...**. Similarly, it is not allowed to over specify the extent of the computational domain. For example, when **h=...** is given, you can not prescribe both **y=...** and **ny=...**.

grid command parameters (part 1)				
Option	Description	Type	Units	Default
x	physical dimension of grid in the x-direction	float	m	none
y	physical dimension of grid in the y-direction	float	m	none
z	physical dimension of grid in the z-direction	float	m	none
h	grid spacing	float	m	none
nx	number of grid points in the x-direction	int	none	none
ny	number of grid points in the y-direction	int	none	none
nz	number of grid points in the z-direction	int	none	none

grid command parameters (geographical coordinates)				
Option	Description	Type	Units	Default
az	clockwise angle from North to the x-axis	float	degrees	135.0
lat	latitude geographical coordinate of the origin	float	degrees	37.0
lon	longitude geographical coordinate of the origin	float	degrees	-118.0

11.1.3 time [required]

Syntax:

time t=... steps=...

Required parameters:

t or steps

The time command specifies the duration of the simulation in seconds or the number of time-steps. The size of the time step is computed internally by *WPP*. You may not over specify the duration of the simulation, i.e., you can not give both **t=...** and **steps=...**

Note that the **prefilter** command changes the start time, see §11.1.5 for a discussion.

time command parameters				
Option	Description	Type	Units	Default
t	duration of simulation	float	s	none
steps	number of cycles (time-steps) to advance	int	none	none

11.1.4 source [required]

Syntax:

source x=... y=... z=... lat=... lon=... depth=... topodepth=... m0=...
 mxx=... mxy=... mxz=... myy=... myz=... mzz=... f0=... fx=... fy=...
 fz=... rake=... strike=... dip=... t0=... freq=... type=... ncyc=...

Required parameters:

See below.

There can be multiple source commands in an input file. Each source command either sets up a point force or a point moment tensor source and should follow the following rules:

- The location of the source must be specified by either a Cartesian location (**x**, **y**, **z**) or by geographical coordinates (**lat**, **lon**) together with (**depth** or **topodepth**). The depth below mean sealevel ($z = 0$) is specified with **depth**, while **topodepth** specifies the depth below the topography.
- Select a point force or a point moment tensor source:
 - Point force: give at least one component of the force vector (**fx**, **fy**, **fz**) and optionally the amplitude **f0**.
 - A point moment tensor source can be specified in one of two ways:
 1. Seismic moment **m0**, and double couple focal mechanism, **strike/dip/rake** angles (see [1]).
 2. At least one component of the moment tensor (**mx**, **my**, etc.) and optionally a scaling factor **m0**.

source command parameters (part 1)				
Option	Description	Type	Units	Default
x	x position of the source	float	m	none
y	y position of the source	float	m	none
z	z position of the source	float	m	none
depth	depth of the source (below $z=0$)	double	m	none
topodepth	depth of the source (below free surface)	double	m	none
lat	latitude geographical coordinate of the source	double	degrees	none
lon	longitude geographical coordinate of the source	double	degrees	none
t0	offset in time	float	s	0.0
freq	frequency	float	Hz or rad/s	1.0
type	Name of time function	string	none	RickerInt
ncyc	Number of cycles (must be specified for the GaussianWindow function)	int	none	none

Options for the time function (**type**) are: **GaussianInt**, **Gaussian**, **RickerInt**, **Ricker**, **Ramp**, **Triangle**, **Sawtooth**, **Smoothwave**, **VerySmoothBump**, **Brune**, **BruneSmoothed**, **GaussianWindow**, and **Liu**. The functions are described in § 4.2.

source command parameters (point moment tensor)				
Option	Description	Type	Units	Default
m0	moment amplitude	float	Nm	1.0
mxx	xx-component of the moment tensor	float	none	0.0
myy	yy-component of the moment tensor	float	none	0.0
mzz	zz-component of the moment tensor	float	none	0.0
mxy	xy-component of the moment tensor	float	none	0.0
mxz	xz-component of the moment tensor	float	none	0.0
myz	yz-component of the moment tensor	float	none	0.0
strike	Aki and Richards strike angle	float	degrees	none
dip	Aki and Richards dip angle	float	degrees	none
rake	Aki and Richards rake angle	float	degrees	none

source command parameters (point force)				
Option	Description	Type	Units	Default
f0	point force amplitude	float	N	1.0
fx	forcing function in the x direction	float	none	0.0
fy	forcing function in the y direction	float	none	0.0
fz	forcing function in the z direction	float	none	0.0

11.1.5 prefilter [optional]

Syntax:

```
prefilter fc=... maxfreq=...
```

Required parameters:

None

The **prefilter** command is used to ensure that the solution is well resolved on the computational grid when the source time functions have high frequency content. The **prefilter** command is particularly useful for computing reliable **hvelmax** and **vvelmax** image files. The **prefilter** command modifies the time functions in all source commands using one or both of the following approaches. If the **maxfreq** parameter is given, the **freq** parameter in all time function is first limited by this value. If the **fc** parameter is given, all source time functions are then filter by a 2-pole 2-pass acausal Butterworth filter. In order to avoid an abrupt start, the minimum threshold value of $\tau_0 = 4/\text{fc}$ is enforced in all source time functions, before the filtering is performed. As a side effect, the start time for the simulation may be negative.

prefilter command parameters				
Option	Description	Type	Units	Default
fc	corner frequency in Butterworth filtering of all source time functions (> 0)	float	Hz	None
maxfreq	Enforce a max threshold value in the freq parameter in all sources (> 0)	float	Hz or rad/s	None

11.2 The material model [required]

It is required to define the material model in the entire computational domain, padded by one layer of ghost cells. However, no material properties need to be given above the topography. The **attenuation** command may be located anywhere in the command file. The material commands **block**, **ifile**, **efile**, and **pfile**, are applied in the same order as they are given. Hence, it is possible to overwrite the properties specified by a material command given earlier in the file. This can be particularly useful when using the **block** command. Finally, the properties of the optional **globalmaterial** command are enforced after reading all other material commands.

11.2.1 attenuation [optional]

The **attenuation** command is used to enable visco-elastic modeling as described in Section 8. The parser scans for the **attenuation** command before reading any of the other material commands, so this command may be located anywhere in the input file. The visco-elastic model uses the quality factors Q_P and Q_S , which may vary from point to point through the computational domain. If visco-elastic modeling is enabled, the Q_P and Q_S factors must be specified as part of every material command described below. When visco-elastic modeling is *not* enabled, the Q_P and Q_S factors are not required in the material commands, and are ignored if present.

Syntax:

attenuation phasefreq=... nmech=... maxfreq=... minppw=...

Required parameters:

None

Note: you may not specify both **maxfreq** and **minppw**.

attenuation command parameters				
Option	Description	Type	Unit	Default
phasefreq	The frequency (> 0) at which V_S and V_P are specified	float	Hz	1.0
nmech	Number of SLS mechanisms to approximate constant Q_P and Q_S (between 1 and 8)	int	None	3
maxfreq	The upper frequency limit (> 0) for approximating constant Q_P and Q_S	float	Hz	2.0
minppw	Calculate the upper frequency limit based on this number of grid points per shortest wave length (> 0)	float	None	None

If you specify the `minppw` option, the upper frequency limit is calculated based on the relation $P = \min V_s/(hf)$, i.e.,

$$f_{max} = \frac{1}{P_{min}} \min \frac{V_s}{h}, \quad f_{max} = \text{maxfreq}, \quad P_{min} = \text{minppw}.$$

11.2.2 block

Syntax:

```
block vp=... vs=... rho=... qp=... qs=... vpgrad=... vsgrad=...
rhograd=... absdepth=... x1=... x2=... y1=... y2=... z1=... z2=...
```

Required parameters:

vp, vs, rho (qp and qs with attenuation)

The `block` command specifies material properties that are constant or vary linearly with depth. By default, the material properties apply to the entire computational domain. By using the optional parameters `x1=...`, `x2=...`, etc., the material properties are only assigned in parts of the computational domain. When used together with the `topography` command, the `absdepth` flag determines how the z -coordinates are used. If `absdepth=0` (default) `z1=...` and `z2=...` specify depths below the free surface. If `absdepth=1`, `z1=...` and `z2=...` bound the z -coordinate of the material block.

The gradient parameters `vpgrad`, `vsgrad`, and `rhograd` specify linear variations in the z -direction (downward). The units for `vpgrad` and `vsgrad` are 1/seconds, which can be interpreted as m/s per m, or km/s per km. The linear variation is relative to the properties at the free surface ($z = 0$ or `depth=0` with `topography`), e.g.,

$$V_p(z) = \mathbf{vp} + z \mathbf{vpgrad}.$$

Note that when `vpgrad` is specified together with `z1 = z1`, $V_p(z_1) = \mathbf{vp} + z_1 \mathbf{vpgrad}$. Hence, the material properties at the top of the block ($z = z_1$) can be very different from `vp` when `z1 vpgrad` is large.

block command parameters				
Option	Description	Type	Units	Default
vp	P-wave velocity	float	m/s	none
vs	S-wave velocity	float	m/s	none
rho	density	float	kg/m ³	none
qp or Qp	P-wave quality factor	float	none	none
qs or Qs	S-wave quality factor	float	none	none
vpgrad	vertical gradient for vp	float	s ⁻¹	none
vsgrad	vertical gradient for vs	float	s ⁻¹	none
rhograd	vertical gradient for rho	float	kg/m ⁴	none
x1	minimum x-dim for the box shaped sub-region	float	m	-max x
x2	maximum x-dim for the box shaped sub-region	float	m	2 max x
y1	minimum y-dim for the box shaped sub-region	float	m	-max y
y2	maximum y-dim for the box shaped sub-region	float	m	2 max y
z1	minimum z-dim for the box shaped sub-region	float	m	-max z
z2	maximum z-dim for the box shaped sub-region	float	m	2 max z
absdepth	z1 and z2 relative to topography (0), or absolute z-coordinate (1)	int	none	0

11.2.3 efile

Syntax:

```
efile etree=... xetree=... logfile=... query=... vsmin=... vpmin=...
access=... resolution=...
```

Required parameters:

```
etree
```

efile command parameters (part 1)				
Option	Description	Type	Units	Default
etree	full path to the etree database file	string	none	none
xetree	full path to the extended etree database file	string	none	none
logfile	name of log file	string	none	none

efile command parameters (part 2)				
Option	Description	Type	Units	Default
vsmin	minimum shear speed V_S	float	m/s	0
vpmin	minimum compresisonal speed V_P	float	m/s	0
query	type of query to perform	string	none	MAXRES
resolution	average properties over this distance (for query=FIXEDRES)	float	m	h
access	can be set to parallel or serial	string	none	parallel

The query option can be set to one of the following:

Query Option	Description
MAXRES	Sample the data at the maximum available resolution in the database. This is the default query type.
FIXEDRES	Average the material properties at the requested resolution, which is specified with the resolution keyword. The default resolution is the grid spacing. h

For example, to set the data to be sampled at 1 km resolution:

```
efile query=FIXEDRES resolution=1000 etree=USGS-SF1906.etree
```

Note: the logfile option can be used to track if any grid points were outside the etree database domain, or if any grid points were located in the air.

11.2.4 pfile

Syntax:

```
pfile filename=... directory=... smoothingsize=... vpmin=... vsmin=...  
rhomin=... flatten=... style=...
```

Required parameters:

filename

pfile command parameters				
Option	Description	Type	Units	Default
filename	name of input pfile	string	none	none
directory	name of directory for the input pfile	string	none	.
smoothingsize	smooth data over stencil of this width	int	none	5
vpmin	minimum threshold value for V_P	float	m/s	0
vsmin	minimum threshold value for V_S	float	m/s	0
rhomin	minimum threshold value for density	float	m/s	0
flatten	Flatten the earth model (T or F)	string	none	F
style	Lat-long or Cartesian grid data	string	none	geographic

11.2.5 ifile

Syntax:

```
ifile filename=...
```

Required parameters:

```
filename
```

The **ifile** command specifies the depth of material surfaces as function of longitude and latitude, and must be used in conjunction with the **material** command. The format for this file is described in Section 12.3, and an example is given in Section 10.3.

ifile command parameters			
Option	Description	Type	Default
filename	name of input file holding material surfaces	string	None

11.2.6 material

Syntax:

```
material id=... vp=... vs=... qp=... qs=... rho=... vpgrad=... vsgrad=...  
rhograd=... vp2=... vs2=... rho2=... vpsqrt=... vssqrt=... rhosqrt=...
```

Required parameters:

```
id, vp, vs, rho (qp and qs with attenuation)
```

The **material** command is used to define material properties together with the **ifile** command, see Section 10.3 for an example.

material command parameters (constants)			
Option	Description	Type	Default
id	material ID number > 0	int	None
vp	P-wave velocity	float	None
vs	S-wave velocity	float	None
rho	Density	float	None
qp or Qp	P-wave quality factor	None	None
qs or Qs	S-wave quality factor	None	None

material command parameters (gradients)			
Option	Description	Type	Default
vpgrad	P-velocity gradient	float	0.0
vsgrad	S-velocity gradient	float	0.0
rhograd	Density gradient	float	0.0

material command parameters (higher order)			
Option	Description	Type	Default
vp2	P-velocity quadratic coefficient	float	0.0
vs2	S-velocity quadratic coefficient	float	0.0
rho2	Density quadratic coefficient	float	0.0
vpsqrt	P-velocity \sqrt{z} coefficient	float	0.0
vssqrt	S-velocity \sqrt{z} coefficient	float	0.0
rhosqrt	Density \sqrt{z} coefficient	float	0.0

11.2.7 globalmaterial [optional]

Syntax:

globalmaterial vpmin=... vsmin=...

Required parameters:

None

The **globalmaterial** command is used to put threshold values on the P - and S -velocities in the material model. These thresholds are enforced after the material properties have been assigned to all grid points.

globalmaterial command parameters			
Option	Description	Type	Default
vpmin	Minimum P-wave velocity (> 0)	float	None
vsmin	Minimum S-wave velocity (> 0)	float	None

11.3 Topography and mesh refinement [optional]

11.3.1 topography [optional]

Syntax:

topography input=... file=... resolution=... zmax=... order=... smooth=...
gaussianAmp=... gaussianXc=... gaussianYc=... gaussianLx=... gaussianLy=...

Required parameters:

input, zmax

Also see discussion below.

The topography command specifies the shape of the free surface boundary, i.e., the vertical extent of the curvilinear grid below the free surface, and optionally the polynomial order of the grid mapping. The topography is given as elevation (in meters) relative to mean sea level, i.e., positive above sea level and negative below sea level. The curvilinear grid is located between the topography and $z = z_{max}$ (recall that z is directed downwards). If the elevation ' e ' of the topography ranges between $e_{min} \leq e \leq e_{max}$, we recommend using $z_{max} \geq -e_{min} + 2|e_{max} - e_{min}|$.

There are four ways of specifying the topography:

- **input=grid** Read the topography as function of latitude and longitude. The file name must be specified by the **file=...** parameter. The format for this file is described in Section 12.1.
- **input=cartesian** Read the topography as function of Cartesian coordinates. The file name must be specified by the **file=...** parameter. The format for this file is described in Section 12.1.
- **input=efile** Read the topography from the Etree data base. The Etree data base must be specified by an **efile** command (see below). The spatial resolution for querying the Etree data base can be specified by the **resolution=...** parameter.
- **input=gaussian** Build an analytical topography in the shape of a Gaussian hill. The amplitude is specified by **gaussianAmp=...**, the hill is centered at **gaussianXc=...**, **gaussianYc=...**, and the half width of the hill in the x and y -directions are specified by **gaussianLx=...**, and **gaussianLy=...**.

topography command parameters (basic)				
Option	Description	Type	Units	Default
input	Type of input: grid, cartesian, efile or gaussian	string	none	none
file	File name if input=grid or input=cartesian	string	none	none
resolution	Resolution for querying the efile if input=efile	float	meters	none
zmax	z coordinate of the interface between Cartesian and curvilinear grid	float	m	0
order	Interpolation order (2, 3 or 4)	int	none	3
smooth	Number of smoothing iterations of topography grid surface	int	none	10

topography command parameters (Gaussian Hill)				
Option	Description	Type	Units	Default
gaussianAmp	Amplitude for a Gaussian hill topography	float	meters	0.05
gaussianXc	x-coordinate of center for a Gaussian Hill	float	meters	0.5
gaussianYc	y-coordinate of center for a Gaussian Hill	float	meters	0.5
gaussianLx	Width of the Gaussian hill in the x-direction	float	meters	0.15
gaussianLy	Width of the Gaussian hill in the y-direction	float	meters	0.15

11.3.2 refinement [optional]

Each **refinement** command corresponds to a mesh refinement patch for $z \leq \mathbf{zmax}$. The grid size in each refinement patch is half of the next coarser grid size. The grid size in the coarsest grid is prescribed by the **grid** command.

Syntax:

```
refinement zmax=...
```

Required parameters:

```
zmax
```

refinement command parameters				
Option	Description	Type	Unit	Default
zmax	maximum z-coordinate for the refinement region	float	m	None

11.4 Output commands [optional]

The output commands enables data to be saved from the simulation. The **sac** command saves a time series of the solution at a recording station, which can be read by the SAC program [6] or the readsac.m Matlab script in the **tools** directory. The **image** command is used to save a two-dimensional cross-section of the solution, the material properties, or the grid. The image files can be read by the readimagepatch.m Matlab script in the **tools** directory. The **volimage** command is used to save three-dimensional volumetric data of the solution, derived quantities of the solution, or the material model. These files are written to a binary format, which can be read by the VisIt post processor, using the **volimage** plug-in. The **gmt** command outputs a shell script file containing the location of all **sac** stations and the epicenter, i.e. the location of the first **source** command. This shell script file can be used for further postprocessing by the GMT program [15].

11.4.1 sac [optional]

The **sac** command is used to save the time history of the solution at a fixed location in space. The **sac** command is described in § 9.2.

Syntax:

```
sac x=... y=... z=... lat=... lon=... depth=... topodepth=... sta=...
file=... type=... writeEvery=... eventDate=... eventTime=... nsew=...
velocity=... usgsformat=... sacformat=... variables=...
```

Required parameters:

Location of the receiver in Cartesian or geographical coordinates.

The file format is described in Section 12.4.

sac command parameters (part 1)				
Option	Description	Type	Units	Default
x	x position of the receiver	float	m	none
y	y position of the receiver	float	m	none
z	z position of the receiver	float	m	none
lat	latitude geographical coordinate of the receiver	float	degrees	none
lon	longitude geographical coordinate of the receiver	float	degrees	none
depth	depth of the receiver (below topography)	float	m	none
topodepth	depth of the receiver (same as depth)	float	m	none

sac command parameters (part 2)				
Option	Description	Type	Units	Default
sta	name of the station	string	none	file
file	file name	string	none	sac
writeEvery	cycle interval to write out the SAC file to disk	int	none	1000
eventDate	date the event occurred: YYYY/MM/DD	int/int/int	none	date of run
eventTime	time the event occurred: hours:minutes:seconds	int:int:int	none	time of run
usgsformat	output all components in an ASCII text file	int	none	0
sacformat	output each component in a SAC file	int	none	1
type	binary or ascii (for SAC format)	string	none	binary
nsew	output (x,y,z)-components (0) or East, North, and vertical ($-z$) components (1)	int	none	0
velocity	output time derivative of solution	int	none	0
variables	solution, curl, divergence, or strains	string	none	solution

11.4.2 image [optional]

Syntax:

```
image x=... y=... z=... time=... timeInterval=... cycle=...
cycleInterval=... file=... mode=... precision=...
```

Required parameters:

Location of the image plane (x, y, or z)

Time for output (time, timeInterval, cycle, or cycleInterval)

Notes:

mode=topo can only be used when the topography command is used.

z=0 corresponds to the free surface when topography is used.

The error in the solution can only be calculated in testing mode, i.e., while using twilight, testlamb, or testpointsource.

The image file format is described in Section 12.5.

image command parameters (part 1)				
Option	Description	Type	Units	Default
x	x location of image plane (≥ 0)	float	m	none
y	y location of image plane (≥ 0)	float	m	none
z	z location of image plane (≥ 0)	float	m	none

image command parameters (part 2)				
Option	Description	Type	Units	Default
time	Time-level for outputting image (closest time step) (≥ 0)	float	s	none
timeInterval	Time-level interval for outputting a series of images (> 0)	float	s	none
cycle	Time-step cycle to output image (≥ 0)	int	none	none
cycleInterval	Time-step cycle interval to output a series of images (≥ 1)	int	none	none
file	File name header of image	string	none	image
precision	Floating point precision for saving data (float or double)	string	none	float
mode	The field to be saved	string	none	rho

mode can take one of the following values:

mode options (grid & geography)	
Value	Description
lat	latitude (in degrees)
lon	longitude (in degrees)
topo	elevation of topography [<i>only available with topography</i>]
grid	grid coordinates in the plane of visualization (<i>e.g.</i> y-z plane if x=const)

mode options (material)	
Value	Description
rho	density
lambda	lambda
mu	mu
p	p velocity
s	s velocity
qp	Q_P quality factor
qs	Q_S quality factor

mode options (solution)	
Value	Description
ux	displacement in the x-direction
uy	displacement in the y-direction
uz	displacement in the z-direction
div	divergence (div) of the displacement
curl	magnitude of the rotation (curl) of the displacement
veldiv	divergence (div) of the velocity
velcurl	magnitude of the rotation (curl) of the velocity
velmag	magnitude of the velocity
hvel	magnitude of the horizontal velocity (North-East components)
hvelmax	maximum in time of the horizontal velocity (North-East components)
vvelmax	maximum in time of the vertical velocity
uxerr	x -component of error (difference between computed and exact solutions)
uyerr	y -component of error (difference between computed and exact solutions)
uzerr	z -component of error (difference between computed and exact solutions)
fx	Forcing in the x-direction
fy	Forcing in the y-direction
fz	Forcing in the z-direction

11.4.3 volimage [optional]

Syntax:

```
volimage file=... mode=... sample=... precision=... savelayer=... cycle=...
cycleInterval=... time=... timeInterval=... startTime=... x1=... x2=...
y1=... y2=... z1=... z2=...
```

Required parameter:

Time for output: (time, timeInterval, cycle, or cycleInterval).

The **volimage** command in *WPP* allows you to save 3D volumetric data on file. These files can be read by the *VisIt* post processor by using a special plug-in, which also is called **volimage**. Be aware that these files can be *very* large. The output occurs at certain time levels during the simulation. The time levels are controlled by the parameters **time=...**, **timeInterval=...**, **cycle=...**, or **cycleInterval=...**, which have the same meaning as in the **image** command. In addition, the **startTime=...** option can be used in conjunction with **cycleInterval** or **timeInterval** to only output data after a specified time level in the simulation. The options **file=...**, **mode=...**, and **precision=...** have the same meaning as the corresponding parameters in the **image** command. The set of possible modes, which is different from the **image** command, is given in the table below. Volimage produces files that are named in the same way as **image** files, but with an added extension **.3D**. before the mode extension, for example, **volimage.cycle=183.3D.curl**. When topography

is present, i.e., the top grid is curvilinear, each **volimage** command produces an additional file with the z -coordinates of the grid. In the above case, this file would be named **volimage.3D.z**. This file is only saved once and is the same for all modes, but depends on the value of **sample**, **savelayer** and **x1**, **x2**, etc. The name of this file is included in the internal header of the corresponding **volimage** solution file. The grid file is needed by *VisIt* to correctly visualize the solution. Note that the grid file is only saved when the input file contains a **topography** command.

By default, when using supergrid far field boundary conditions, the sponge layer is removed from the volimage data set. The parameter option **savelayer=yes** forces WPP to save the complete field, including the sponge layers. By default, the solution is saved at every grid point. The size of the data file can be significantly reduced by using the **sample** option. For example, **sample=2** only saves the solution at every other grid point, **sample=3** at every third point, etc. The coarsening of the grid is the same in all three space dimensions. The size of the file can also be reduced by using the optional parameters **x1=...**, **x2=...**, etc.. In this case, only the specified sub-domain is saved to file.

volimage command options (part 1)				
Option	Description	Type	Units	Default
file	file name header of image	string	none	volimage
mode	specifies which field is written to the image file	string	none	rho
sample	save only every sample:th grid point	int	none	1
precision	precision of image data on file (float/double)	string	none	float
savelayer	Include sponge layer at far field boundaries in saved data (yes/no)	string	none	no

volimage command options (part 2)				
Option	Description	Type	Units	Default
time	simulation time to output image, will be closest depending on dt taken	float	s	none
timeInterval	simulation time interval to output series of images	float	s	none
cycle	time-step cycle to output image	int	none	none
cycleInterval	time-step cycle interval to output a series of images	int	none	none
startTime	only output data after this time level (only used with cycleInterval or timeInterval)	float	s	-999.9
x1	minimum x-dim for the box shaped sub-region	float	m	0
x2	maximum x-dim for the box shaped sub-region	float	m	max x
y1	minimum y-dim for the box shaped sub-region	float	m	0
y2	maximum y-dim for the box shaped sub-region	float	m	max y
z1	minimum z-dim for the box shaped sub-region	float	m	topography
z2	maximum z-dim for the box shaped sub-region	float	m	max z

Options for mode include:

volimage mode sub-options	
Value	Description
ux	displacement in the x-direction
uy	displacement in the y-direction
uz	displacement in the z-direction
rho	density
p	p velocity
s	s velocity
div	divergence (div) of the displacement
curl	magnitude of the rotation (curl) of the displacement
veldiv	divergence (div) of the velocity
velcurl	magnitude of the rotation (curl) of the velocity
mag	magnitude of the displacement
velmag	magnitude of the velocity

Note that a supplemental **volimage** file (with extensions **.z**) is created when topography is used. This file, which contains the *z*-coordinates of the grid, is automatically read by the **volimage** plug-in for *VisIt* and allows it to construct the computational grid.

11.4.4 gmt [optional]

Syntax:

gmt file=...

Required parameters:

None.

gmt command parameters			
Option	Description	Type	Default
file	name of output file for gmt c-shell commands	string	wpp.gmt.csh

11.5 WPP testing commands [optional]

11.5.1 twilight

The **twilight** command runs *WPP* in a testing mode where forcing functions are constructed to create a known smooth analytical solution, see Appendix B.1 for details.

Syntax:

twilight errorlog=... omega=... c=... phase=... momega=... mphase=...
amprho=... ampmu=... amplambda=...

Required parameters:

None

twilight command parameters			
Option	Description	Type	Default
errorlog	Outputs error log in file twilight_errors.dat	int	0
omega	Wave number in exact solution	float	1.0
c	Phase speed in exact solution	float	1.3
phase	Solution phase coefficient	float	0.0
momega	Wave number in material	float	1.0
mphase	Material phase coefficient	float	0.4
amprho	Density amplitude	float	1.0
ampmu	Material μ amplitude	float	1.0
amplambda	Material λ amplitude	float	1.0

11.5.2 testlamb

The **testlamb** command solves Lamb's problem, i.e., the displacement due to a vertical point forcing on a flat free surface, see Appendix B.2 for details.

Syntax:

```
testlamb x=... y=... cp=... rho=... fz=...
```

Required parameters:

Location of the forcing (x, y) .

testlamb command parameters			
Option	Description	Type	Default
x	x-coordinate of point source	float	0.0
y	y-coordinate of point source	float	0.0
cp	P-wave velocity	float	1.0
rho	Density	float	1.0
fz	Magnitude of the forcing	float	1.0

11.5.3 testpointsource

The **testpointsource** command calculates the displacement due to a point source in a homogeneous whole space, and computes the error. Note that the reported errors are only reliable before the solution has reached the outflow boundaries. Look in the source code for further information.

Syntax:

```
testpointsource x=... y=... z=... cp=... cs=... rho=... m0=... mxx=...
mxy=... mxz=... myy=... myz=... mzz=... f0=... fx=... fy=... fz=...
freq=... t0=... type=...
```

Required parameters:

None

testpointsource command parameters (part 1)			
Option	Description	Type	Default
x	x-coordinate of point source	float	0
y	y-coordinate of point source	float	0
z	z-coordinate of point source	float	0
freq	Frequency of the forcing	float	1
t0	Offset in time	float	1
type	Type of the source: SmoothWave, VerySmooth-Bump,Ricker	string	Ricker
cp	P-wave velocity	float	$\sqrt{3}$
cs	S-wave velocity	float	1
rho	Density	float	1

testpointsource command parameters (point source type)			
Option	Description	Type	Default
m0	Moment amplitude	float	1
mxx	xx-component of moment tensor	float	0
mxy	xy-component of moment tensor	float	0
mxz	xz-component of moment tensor	float	0
myy	yy-component of moment tensor	float	0
myz	yz-component of moment tensor	float	0
mzz	zz-component of moment tensor	float	0
f0	Point force amplitude	float	1
fx	Magnitude of the forcing in the x-direction	float	1
fy	Magnitude of the forcing in the y-direction	float	1
fz	Magnitude of the forcing in the z-direction	float	1

11.6 Advanced simulation controls [optional]

WARNING! The commands in this section are only intended for advanced users who are intimately familiar with the inner workings of *WPP*. These commands might lead to unexpected side effects. Only the source code gives a complete description of what these commands really do.

11.6.1 supergrid [optional]

Syntax:

`supergrid thickness=... damping_coefficient=...`

Required parameters:

None

supergrid command parameters			
Option	Description	Type	Default
thickness	Thickness of the supergrid region	float	15 h
damping_coefficient	Damping coefficient in supergrid region	float	0.15

11.6.2 boundary_conditions [optional]

Syntax:

`boundary_conditions lx=... hx=... ly=... hy=... lz=... hz=...`

Required parameters:

None

Boundary conditions parameters			
Option	Description	Type	Default
lx	Boundary condition at $x = 0$	int 0-5	5
hx	Boundary condition at $x = x_{max}$	int 0-5	5
ly	Boundary condition at $y = 0$	int 0-5	5
hy	Boundary condition at $y = y_{max}$	int 0-5	5
lz	Boundary condition at $depth = 0$	int 0-5	2
hz	Boundary condition at $z = z_{max}$	int 0-5	5

boundary condition Type values	
Value	Type
0	Clayton-Enquist boundary
1	Energy absorbing boundary
2	Stress-free boundary
3	Dirichlet boundary
4	Neumann boundary
5	Supergrid boundary

11.6.3 developer [optional]

Warning: you need to be intimately familiar with the inner workings of *WPP* to use this command. Look in the source code to get a full understanding of what this command really does.

Syntax:

```
developer cfl_number=... interpolation=... ctol=... cmaxit=... output_load=...
output_timing=... log_energy=... print_energy=... mpiio=... iotiming=...
```

Required parameters:

None

developer parameters (part 1)			
Option	Description	Type	Default
cfl_number	CFL number (> 0)	float	0.8
interpolation	Interpolation type at grid refinement boundaries (conservative or non-conservative)	string	conservative
ctol	Relative tolerance for iterative solution of conservative grid refinement (> 0)	float	1e-3
cmaxit	Max number of iterations for solving conservative grid refinement equations (> 0)	int	20

developer parameters (part 2)			
Option	Description	Type	Default
output_load	Output load info (0 or 1)	int	0
output_timing	Output timing info (0 or 1)	int	0
log_energy	File name for saving energy info	string	none
print_energy	Save energy information (0 or 1)	int	0
mpiio	Use the standard MPI-I/O (1) or Bjorn's fast I/O (0) routines for saving image files	int	0
iotiming	output timing info after each image is written to disk. (0 or 1)	int	0

Chapter 12

File formats

12.1 topography

Topography is specified as elevation above mean sea level on a regular lattice in the horizontal plane. There are two variants of the topography format: geographic or Cartesian. By default, topography is specified as function of geographic coordinates in the horizontal plane. Alternatively, the lattice can be specified in Cartesian coordinates. In both cases, the unit for elevation is meters and the topography file must cover the entire horizontal extent of the computational domain.

12.1.1 topography on a geographic lattice

Latitude and longitude should be given in degrees. Let the elevation be known at longitudes

$$\phi_i, \quad i = 1, 2, \dots, N_{lon},$$

and latitudes

$$\theta_j, \quad j = 1, 2, \dots, N_{lat},$$

Note that the latitudes and the longitudes must either be strictly increasing or strictly decreasing, but the step size may vary.

The elevation should be given on the regular lattice

$$e_{i,j} = \text{elevation at longitude } \phi_i, \text{ latitude } \theta_j.$$

Bi-cubic interpolation is used to define the elevation in between the lattice points.

The topography file should be an ASCII text file with the following format. The first line of the file holds the number of longitude and latitude data points:

$$N_{lon} \quad N_{lat}$$

On subsequent lines, longitude, latitude and elevation values are given in column first ordering:

$$\begin{array}{ccc}
 \phi_1 & \theta_1 & e_{1,1} \\
 \phi_2 & \theta_1 & e_{2,1} \\
 \vdots & \vdots & \vdots \\
 \phi_{Nlon} & \theta_1 & e_{Nlon,1} \\
 \vdots & \vdots & \vdots \\
 \phi_1 & \theta_{Nlat} & e_{1,Nlat} \\
 \phi_2 & \theta_{Nlat} & e_{2,Nlat} \\
 \vdots & \vdots & \vdots \\
 \phi_{Nlon} & \theta_{Nlat} & e_{Nlon,Nlat}
 \end{array}$$

12.1.2 topography on a Cartesian lattice

Cartesian coordinates should be given in meters ([m]). Let the elevation be known at x -coordinates

$$x_i, \quad i = 1, 2, \dots, Nx,$$

and y -coordinates

$$y_j, \quad j = 1, 2, \dots, Ny,$$

Note that the coordinate vectors must either be strictly increasing or strictly decreasing, but the step size may vary. Also note that the step size can be different from the step size in the computational grid. To guarantee that the topography grid covers the entire horizontal extent of the computational domain, we require

$$\min_i x_i \leq 0, \quad \min_j y_j \leq 0, \quad \max_i x_i \geq x_{max}, \quad \max_j y_j \geq y_{max},$$

where x_{max} and y_{max} are defined by Equation (3.1). Bi-cubic interpolation is used to define the elevation in between the lattice points.

The elevation should be given on the regular lattice

$$e_{i,j} = \text{elevation at Cartesian coordinate } (x, y) = (x_i, y_j).$$

The topography file should be an ASCII text file with the following format. The first line of the file holds the number of data points in each direction:

$$Nx \quad Ny$$

On subsequent lines, x , y and elevation values are given in column first ordering:

$$\begin{array}{ccc}
 x_1 & y_1 & e_{1,1} \\
 x_2 & y_1 & e_{2,1} \\
 \vdots & \vdots & \vdots \\
 x_{Nx} & y_1 & e_{Nx,1} \\
 \vdots & \vdots & \vdots \\
 x_1 & y_{Ny} & e_{1,Ny} \\
 x_2 & y_{Ny} & e_{2,Ny} \\
 \vdots & \vdots & \vdots \\
 x_{Nx} & y_{Ny} & e_{Nx,Ny}
 \end{array}$$

12.2 pfile

There are two variants of the pfile format: geographic or Cartesian. By default, geographic coordinates are used to specify the location of the depth profiles in the horizontal plane. Alternatively, the lattice can be specified in Cartesian coordinates. Note that different units are used in the two cases.

12.2.1 pfile on a geographic lattice

The header has 7 lines and follows the following format:

Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	Δ [deg] (real)			
3	N_{lat} (integer)	Lat_{min} [deg] (real)	Lat_{max} [deg] (real)	
4	N_{lon} (integer)	Lon_{min} [deg] (real)	Lon_{max} [deg] (real)	
5	N_{dep} (integer)	d_{min} [km] (real)	d_{max} [km] (real)	
6	I_{sed} (integer)	I_{MoHo} (integer)	I_{410} (integer)	I_{660} (integer)
7	Q -available? (logical)			

Lines 3 and 4 contain the number of lattice points as well as the starting and ending angles in the latitude and longitude direction, respectively, . Line 5 contains the number of depth values in each profile, followed by the minimum and maximum depth measured in km. Line 6 supplies optional information about the index of some material discontinuities in each depth profile. Give -99 if not known. Note that the index for each discontinuity (sediment, MoHo, 410, 660) indicates the row number within each profile, for the material property just above the discontinuity. Hence, the subsequent entry in each profile should have the same depth value and contain the material property just below the same discontinuity. Line 7 should contain the single letter 'T' or 't' if the subsequent data contains quality factors (Q_P and Q_S); otherwise it should contain the single

letter 'F' or 'f'. The presence of quality factors may alternatively be indicated by using the strings '.TRUE.', '.true.', '.FALSE.', or '.false.'.

The first seven lines of a pfile can look like this:

```
Caucasus
0.25
7 38.00 39.50
19 44.50 49.00
30 0.00 161.00
-99 -99 -99 -99
.TRUE.
```

The header is directly followed by $N_{lat} \times N_{lon}$ depth profiles, ordered such that the longitude varies the fastest, that is, according to the pseudo-code:

```
for ( $Lat_i = Lat_{min}$ ;  $Lat_i \leq Lat_{max}$ ;  $Lat_i += \Delta$ )
  for ( $Lon_j = Lon_{min}$ ;  $Lon_j \leq Lon_{max}$ ;  $Lon_j += \Delta$ )
    (save depth profile for  $Lat_i, Lon_j$ )
  end
end
```

The first line of each depth profile holds the latitude and longitude (in degrees as real numbers), and the number of depth values, which must equal N_{dep} . For example a depth profile for latitude 33.108, longitude -115.66, with $N_{dep} = 19$ points in the depth direction starts with the line

```
33.108 -115.66 19
```

The subsequent N_{dep} lines have the following format:

Index (int)	depth [km]	V_p [km/s]	V_s [km/s]	ρ [g/cm ³]	Q_P	Q_S
-------------	------------	--------------	--------------	-----------------------------	-------	-------

Note that Q_P and Q_S should only be present when indicated so by the Q -availability flag on line 7 of the header. Also note that the units are different than in other parts of *WPP*. In particular, V_P and V_S should be given in km/s= 1000 m/s, and density (ρ) should be given in g/cm³ = 1000 kg/m³.

12.2.2 pfile on a Cartesian lattice

The header of the Cartesian grid pfile format consists of seven lines with the following information:

Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	h [m] (real)			
3	N_x (integer)	x_{min} [m] (real)	x_{max} [m] (real)	
4	N_y (integer)	y_{min} [m] (real)	y_{max} [m] (real)	
5	N_{dep} (integer)	d_{min} [m] (real)	d_{max} [m] (real)	
6	I_{sed} (integer)	I_{MoHo} (integer)	I_{410} (integer)	I_{660} (integer)
7	Q -available? (logical)			

This is the same header as for the geographic coordinate format, with the only difference that information on lines 2, 3, and 4 is different. The spacing, h , of the grid of depth profiles is given on line 2. The number of depth profiles in the x -direction N_x with minimum and maximum coordinate values are given on line 3. The same quantities for the y -direction are given on line 4. Note that all distances, including the depth information on line 5, must be given in meters ($[m]$).

The header is directly followed by $N_x \times N_y$ depth profiles, ordered such that the x -coordinate varies the fastest, that is, according to the pseudo-code:

```

for ( $y = y_{min}; y \leq y_{max}; y+ = h$ )
  for ( $x = x_{min}; x \leq x_{max}; x+ = h$ )
    (save depth profile for  $x, y$ )
  end
end
end

```

The first line of each depth profile holds the x -coordinate and the y -coordinate (in meters as real numbers), and the number of depth values, which must equal N_{dep} . For example a depth profile for $x = 100.4 m$ and $y = 30.6 m$, with $N_{dep} = 19$ points in the depth direction starts with the line

100.4 30.6 19

The subsequent N_{dep} lines have the following format:

Index (int)	depth [m]	V_p [m/s]	V_s [m/s]	ρ [kg/m ³]	Q_P	Q_S
-------------	-----------	-------------	-------------	-----------------------------	-------	-------

Q_P and Q_S can be left out when indicated not present by the Q -availability flag on line 7 of the header. Note that, unlike the pfiles on a geographic lattice, the units should here be the standard MKS units, which normally are used in *WPP*.

12.3 ifile

The material surface file (ifile) should be an ASCII text file with the following format. The first line of the file holds the number of longitude and latitude data points, as well as the number of material surfaces:

$$N_{lon} \quad N_{lat} \quad N_{mat}$$

On subsequent lines, longitude, latitude and N_{mat} surface depth values are given in column first ordering:

$$\begin{array}{ccccc}
 Lon_1 & Lat_1 & d_{1,1,1} & \dots & d_{N_{mat},1,1} \\
 Lon_2 & Lat_1 & d_{1,2,1} & \dots & d_{N_{mat},2,1} \\
 \vdots & \vdots & \vdots & & \vdots \\
 Lon_{N_{lon}} & Lat_1 & d_{1,N_{lon},1} & \dots & d_{N_{mat},N_{lon},1} \\
 \vdots & \vdots & \vdots & & \vdots \\
 Lon_1 & Lat_{N_{lat}} & d_{1,1,N_{lat}} & \dots & d_{N_{mat},1,N_{lat}} \\
 Lon_2 & Lat_{N_{lat}} & d_{1,2,N_{lat}} & \dots & d_{N_{mat},2,N_{lat}} \\
 \vdots & \vdots & \vdots & & \vdots \\
 Lon_{N_{lon}} & Lat_{N_{lat}} & d_{1,N_{lon},N_{lat}} & \dots & d_{N_{mat},N_{lon},N_{lat}}
 \end{array}$$

It is required that $d_{q,i,j} \leq d_{q+1,i,j}$.

12.4 sac

SAC files hold the time history of one component of the solution at a fixed point in space. A detailed description of the SAC format can be found at <http://www.iris.edu/manuals/sac/manual.html>. In the `tools` directory, we provide a simplified Matlab reader of SAC files called `readsac.m`. Note that only some of the header information is parsed by this reader:

```
% READSAC
%
%   Read SAC receiver data.
%
%   [u,dt,lat,lon,t0] = readsac( fname, format )
%
%       Input: fname - Name of SAC file
%              format - Little endian ('l') or big endian ('b')
%                      byte order for binary data. Default is 'l'.
%
%       Output: u      - The data component on SAC file
%              dt      - Uniform time step for u
%              lat, lon - Lat and Lon of the SAC station.
%              t0      - Start time for time-series
%
function [u,dt,lat,lon,t0] = readsac( fname, format )
if nargin < 2
    format = 'l';
end;

fid = fopen(fname,'r',format);
if fid < 0
    disp( ['Error: could not open file ' fname] );
else
    dt = fread( fid,1,'float32');
    fseek(fid,4*4,0);
    t0 = fread( fid,1,'float32');
    fseek(fid,25*4,0);
    lat = fread(fid,1,'float32');
    lon = fread(fid,1,'float32');
    fseek(fid,2*4,0);
    evlat = fread(fid,1,'float32');
    evlon = fread(fid,1,'float32');
    fseek(fid,4,0);
    evdepth = fread(fid,1,'float32');
    disp(['Begin time (t0) = ' num2str(t0)]);
    disp(['Event lat lon = ' num2str(evlat) ' ' num2str(evlon) ]);
    disp(['Event depth ' num2str(evdepth) ' km']);
    fseek(fid,4*40,0);
    npts=fread(fid,1,'int');
```

```

    fseek(fid,78*4,0);
    u=fread(fid,npts,'float32');
    fclose(fid);
end

```

12.5 image

Images files hold two-dimensional data on a composite grid and are written in a binary format. The header of the file starts with two integers: the precision (4 for single precision, 8 for double precision), and the number of patches. After that follows header info for each patch, consisting of the grid size h (a double precision floating point number) and four integers holding the starting and ending indices for each patch. The header is followed by the two-dimensional data on each patch, consisting of one single or double precision floating point number for each grid point, stored in column-first order.

The exact format follows from the Matlab function `tools/readimagepatch.m` which is provided in the source distribution of *WPP*:

```

% Returns image patch nr. 'inr' on file 'fil' in 'im',
% corresponding grid returned in 'x' and 'y'
function [im,x,y]=readimagepatch( fil, inr )

fd=fopen(fil,'r');

% Precision of image data (4-float, 8-double)
pr=fread(fd,1,'int');

% Number of image patches on file
ni=fread(fd,1,'int');
if inr > ni
    disp( 'Error image nr too large' );
else
% For each patch read grid spacing and index bounds.
% For patch nr. p:  ib(p) <= i <= ie(p) and jb(p) <= j <= je(p)
    for i=1:ni
        h(i) = fread(fd,1,'double');
        ib(i) = fread(fd,1,'int');
        ie(i) = fread(fd,1,'int');
        jb(i) = fread(fd,1,'int');
        je(i) = fread(fd,1,'int');
    end;
% Want patch nr. inr, skip the first inr-1 image patches.
    for i=1:inr-1
        fseek(fd,(ie(i)-ib(i)+1)*(je(i)-jb(i)+1)*pr,0);
    end;
% Read wanted image patch, single or double precision.
    if pr == 4

```

```

        im = fread(fd,[ie(inr)-ib(inr)+1 je(inr)-jb(inr)+1],'float');
    else
        im = fread(fd,[ie(inr)-ib(inr)+1 je(inr)-jb(inr)+1],'double');
    end;
% Corresponding Cartesian grid
    x = ((ib(inr):ie(inr))-1)*h(inr);
    y = ((jb(inr):je(inr))-1)*h(inr);
    fclose(fd);
% transpose im and return result
    im = im';
end;

```

In this implementation, `fd` is a file descriptor variable. The Matlab functions `fopen` and `fread` perform binary I/O similarly to the C functions with the same names.

Note that the above matlab function reads one image patch from an image file into the Matlab matrix `im`. The corresponding Cartesian coordinates are returned in the Matlab vectors `x` and `y`.

12.6 volimage

The `volimage` command generates (often very large) binary files holding three-dimensional volumetric data. Most users will visualize the data in these files with the open source *VisIt* post processor. For this reason, we will not describe the binary format here. If you really want to know the details of the format, we suggest you look in the source code for the plug-in (in the `wpp/tools/visit/volimage` directory).

Starting with version 2.4 of *VisIt*, the `volimage` plug-in for reading these files should be included with the official release of *VisIt*. If you see the `volimage` format in the drop-down file format menu in *VisIt*, there is no need to build a local version of the plug-in.

In this section, we provide instructions on how to build the `volimage` plug-in in case it is not available in your version of *VisIt*. The source code for the plug-in is included with the *WPP* source code distribution in the directory `tools/visit/volimage`. On the Livermore Computing (LC) machines, the source code can be found in the directory `/usr/apps/wpp/tools/visit/volimage`. In order to use this plug-in on the LC machines, you must first copy the source code to a local directory where you have write privileges.

To build the `volimage` plugin, you do the following:

1. Add *VisIt* to your search path by editing the `PATH` environmental variable. For example, if you are using C-shell on one of the LC machines, you should edit your `~/.cshrc` file and add the directory `/usr/gapps/visit/bin` to `PATH`. Next time you open a shell, verify that you have *VisIt* in your search path by issuing the command `which visit`.
2. Make sure you have access to the `cmake` command, version ≥ 2.8 . Do `cmake --help` to check the version. On LC, newer versions of `cmake` can be found under `/usr/gapps/visit/cmake`.
3. If you don't have write privileges on `tools/visit/volimage`, copy `tools/visit/volimage` to a local directory (edit `MYDIR`). Then go to that directory:

```

cp ....tools/visit/volimage/* MYDIR/.
cd MYDIR/volimage

```

You build the plug-in using the following commands:

4. `xml2cmake -clobber volimage.xml`
5. `cmake .` (note the '.')
6. `make`

The "make" step will compile the plugin libraries and put them into the directory

`~/visit/<platform>/plugins/databases`

To use the `volimage` plugin, you first need to run *WPP* to create a volimage data file. Sample input files for *WPP* can be found in the `wpp/examples/volimage` directory. On LC, these files are in the directory `/usr/apps/wpp/examples/volimage`. Once a `volimage` file has been generated, you can visualize it by running *VisIt* and open the data file using the `volimage` format, which should be available from the file format pull-down menu.

Chapter 13

Run time and memory requirements

13.1 Run time

The execution times shown in Table 13.1 were obtained by running WPP on a problem with 0.5-1 M grid points per processor. Timings were measured on 16 processors of the sierra machine at LLNL, in July of 2011. The WPP executable was compiled with Intel compilers at optimization level -O. While the absolute numbers in Table 13.1 are machine dependent and destined to change in the (near) future, the relative cost of various solver configurations is likely to stay more or less constant. Note that the visco-elastic cases used 3 mechanisms. For the case with topography, about 82% of the grid points were in the curvilinear grid.

We note that 1): solving the visco-elastic wave equations with $m=3$, requires a factor 3.7 more CPU time than the purely elastic case. 2): Satisfying the interface conditions for grid refinement is relatively expensive. With one refinement and the elastic solver, those conditions required approximately 60% of the total computation time. For the visco-elastic case, more calculations per grid point are performed in the interior of the domain, and the relative cost for enforcing the interface conditions goes down to about 35% of the total time. 3): The scaling is not perfect and increasing the number of processors, while keeping the number of grid points constant, might lead to some degradation in the performance. More experimentation is needed to better evaluate the strong scaling properties of WPP.

13.2 Memory usage

The memory usage in Table 13.2 was obtained by running WPP on a single processor using 1-2 M grid points. By inspection of the source code, we found the following number of 3-D arrays holding double precision (8 bytes) floating point variables: 15 for Cartesian grids, another 13 for topography (coordinates and metric coefficients in the curvilinear grid), and an additional $5+11*m$ for a m -mechanism visco-elastic material. These numbers correspond to the asymptotic memory requirements presented in the right column of Table 13.2, which agree well with the observed memory usage. The observed numbers tend to the theoretical asymptotic numbers as the problem size increases.

As expected, the observed memory usage closely follows the linear relation

$$M = k * N + m,$$

Configuration	Solver	Execution time [s]	Relative factor
Cartesian, single grid	elastic	$6.90 \cdot 10^{-9}$	1
Cartesian, one refinement	elastic	$1.64 \cdot 10^{-8}$	2.4
Cartesian, two refinements	elastic	$1.77 \cdot 10^{-8}$	2.7
Topography	elastic	$3.48 \cdot 10^{-8}$	5.0
Cartesian, single grid	visco-elastic	$2.57 \cdot 10^{-8}$	3.7
Cartesian, one refinement	visco-elastic	$3.49 \cdot 10^{-8}$	5.1
Topography	visco-elastic	$1.27 \cdot 10^{-7}$	18.0

Table 13.1: Execution time per time step and grid point for different grid configurations, both for solving the elastic and the visco-elastic wave equations. The relative factor is the execution time relative to the Cartesian single grid case, for the elastic wave equation.

Configuration	solver	Memory [bytes/grid point]	Asymp. usage [bytes/grid point]
Cartesian grid	elastic	135	120
Topography	elastic	228	205
Cartesian grid	visco-elastic	440	424
Topography	visco-elastic	530	509

Table 13.2: Memory usage for different grid and solver configurations. The visco-elastic case has 3 mechanisms and the case with topography has 82% of the grid point in the curvilinear grid.

where M is memory usage and N is the number of grid points. For parallel runs, the memory per grid point will be somewhat higher, because of duplicated ghost points at the processor boundaries. Here we only report the memory usage for a single processor run. Note that the `volimage` command saves a copy of the image variable over the full 3D volume. This adds (at single precision, and full resolution) 4 bytes per grid point and per volume image command. It is possible to reduce this size by only saving every n :th grid point in each coordinate direction. For example, only saving every 2nd point reduces the size of the data by a factor 8.

Appendix A

Installing *WPP*

The *WPP* source code is released under the GNU general public license and can be downloaded from:

<https://computation.llnl.gov/casc/serpentine/software.html>

Version 2.2 of *WPP* is built using `make`. We recommend using GNU `make`, sometimes called `gmake`. You can check the version of `make` on your system with the command

```
shell> make -v
```

If you don't have GNU `make` installed, you can obtain it from www.gnu.org.

A.1 Supported platforms and compilers

We have built *WPP* and its supporting libraries on Intel based desktops and laptops running LINUX and OSX. It has also been built on various supercomputers such as the large Linux clusters at LLNL (currently zeus, atlas and sierra), as well as on IBM BG/L and BG/P systems. We have built *WPP* using Gnu, Intel, or IBM compilers. Our experience is that *WPP* is likely to build if the underlying third party libraries can be built. Currently, we are using the following compiler versions:

Gnu:	<code>g++/gcc/gfortran</code>	versions 4.3 to 4.5
Intel:	<code>icpc/icc/ifort</code>	versions 9.1 to 11.1
IBM:		version info currently unavailable

A.2 MPI and other third party libraries

Before you can build *WPP*, you must install the third party libraries. All libraries need to be installed in the same directory, such that each library installs its files in the `lib` and include sub-directories. To avoid incompatibility issues and linking problems, it is advisable to use the same compiler for building the third party libraries as for building *WPP*.

For a basic installation of *WPP*, you need:

1. the lapack and blas libraries, which provide basic linear algebra functionality. On many machines these libraries are preinstalled,

2. an MPI-2 library, which provides support for message passing on parallel machines. An example is the mpich2 library. Note that you need the MPI-2 library even when installing WPP on a single core system.

Note that the MPI-2 library often include wrappers for compiling, linking, and executing MPI programs. For example, the mpich2 package includes the mpicxx and mpif77 compilers, as well as the mpirun script. These wrappers simplify the compilation and linking process and we highly recommend using them.

For a complete installation that supports the efile command for reading e-tree material data bases, you need to install the following additional libraries (see Section A.5 for details):

- The Etree library
- The Proj library
- The cencalvm library

A.2.1 Mac computers

We recommend using the MacPorts package manager for installing the required compilers and libraries. Simply go to www.macports.org, and install macports on your system. With that in place, you can use the `port` command as follows

```
shell> port install gcc44
shell> port install mpich2 +gcc44
```

Here, gcc44 refers to version 4.4 of the Gnu compiler suite. Compiler versions are bound to change with time. Before starting, make sure you install a version of gcc that is compatible with the mpich2 package. By adding the gcc44 variant, you also get a compatible Fortran compiler.

The lapack and blas libraries are preinstalled on recent Macs and can be accessed using the “-framework vecLib” link option. If that does not work on your machine, you can download those libraries from www.netlib.org.

A.2.2 Linux machines

We here give detailed instructions for installing the third part libraries under 64 bit, Fedora Core 14 Linux. Other Linux variants use similar commands for installing software packages.

You need to have root privileges to install precompiled packages. Start by opening an xterm as root using the command

```
su -
```

Install the compilers by issuing the commands

```
yum install gcc
yum install gcc-c++
yum install gcc-gfortran
```

You install the mpich2 library and include files with the command

```
yum install mpich2-devel
```

The `blas` and `lapack` libraries are installed with

```
yum install blas
yum install lapack
```

On our system, the libraries were installed in `/usr/lib64` as `libblas.so.3` and `liblapack.so.3`. For some unknown reason, the install program does not add links to these files with extension `.so`, which is necessary for the linker to find them. We must therefore add the links explicitly. If the libraries were installed elsewhere on your system, but you don't know where, this is how to find them: Give the commands:

```
cd /
find . -name "*blas*" -print
```

After locating the libraries, we add links to the libraries by issuing the commands (you need to be `root` for this to work)

```
cd /usr/lib64
ln -s libblas.so.3 libblas.so
ln -s liblapack.so.3 liblapack.so
```

A.3 Directory structure

To unpack the *WPP* source code, you place the file `wpp-version-2.2.tar.gz` in the desired location and issue the following commands:

```
shell> gunzip wpp-version-2.2.tar.gz
shell> tar xvf wpp-version-2.2.tar
```

Afterwards, you will find a new directory named `wpp-version-2.2`, which contains several files and subdirectories:

- `LICENSE.txt` License information.
- `INSTALL.txt` Information about how to build *WPP* (short version).
- `KNOWN-BUGS.txt` List of known problems, porting issues, or bugs.
- `README.txt` General information about *WPP*.
- `configs` Directory containing `make` configuration files.
- `src` C++ and Fortran source code of *WPP*.
- `tools` Matlab scripts for post processing and analysis.
- `examples` Sample input files.
- `Makefile` Main makefile (don't change this file!).
- `wave.txt` Text for printing the "WPP Lives" banner at the end of a successful build.

A.4 Compiling and Linking *WPP* (without the *cencalvm* library)

The best way of getting started is to first build *WPP* without the *cencalvm* library. This process should be very straight forward and the resulting *WPP* executable supports all commands except the `efile` command. If you need to use the `efile` command, you can later install the *cencalvm* and supporting libraries, see § A.5, and then recompile *WPP*.

Start by familiarizing yourself with the *wpp* source code by listing the main *wpp* directory,

```
shell> cd /enter/your/path/wpp-version-2.2
shell> ls
```

Go into the `configs` directory:

```
shell> cd configs
```

You should set the environment variable `WPPMAKE` to point to a configuration file for the main "Makefile". Examples for some common operating systems are given in the `configs` directory (`make.amac` for Mac's; `make.tux`, `make.fc14`, and `make.sierra` for Linux machines). For example, to use the `configs/make.amac` setup, you can use the `setenv`-command

```
setenv WPPMAKE /enter/your/path/wpp-version-2.2/configs/make.amac
```

If you are using a Bourne, `bash`, or related shell, you should instead use the command

```
export WPPMAKE=/enter/your/path/wpp-version-2.2/configs/make.amac
```

It is a good idea to include this command in your setup file (`.cshrc` for `c-shell`).

You build *WPP* with the "make" command from the main directory.

```
shell> cd /enter/your/path/wpp-version-2.2
shell> make
```

If you for some reason wish to remove all object files and the *wpp* executable, you can do

```
shell> cd /enter/your/path/wpp-version-2.2
shell> make clean
```

By default, the *WPP* executable is located in

```
/my/installation/dir/wpp-version-2.2/optimize_v2.2/wpp
```

It can be convenient to add this directory to your `PATH` environment variable (i.e., modify your `.cshrc` file).

You can also build a debug version of *WPP* by adding the `debug=1` option to `make`,

```
shell> cd /my/installation/dir/wpp-version-2.2
shell> make debug=1
```

That executable will be located in

```
/my/installation/dir/wpp-version-2.2/debug_v2.2/wpp
```

A.4.1 How do I setup the make.inc file?

The main input file for make is

wpp-version-2.2/Makefile

Do *not* change the Makefile. It should only be necessary to edit your configuration file (the file name should be given in the WPPMAKE environment variable), for example

/my/path/wpp-version-2.2/configs/make.inc

This file holds all information that is particular for your system, such as the name of the compilers, the location of the third party libraries, and any extra arguments that should be passed to the compiler or linker. This file also instructs make whether or not the cencalvm library should be linked to WPP.

The following file includes all configurable options:

```
etree = no
```

```
CXX = mpicxx
```

```
CC = mpicc
```

```
FC = mpif77
```

```
WPPROOT = /Users/petersson1
```

```
EXTRA_CXX_FLAGS = -DUSING_MPI
```

```
EXTRA_FORT_FLAGS = -fno-underscoring
```

```
EXTRA_LINK_FLAGS = -framework vecLib
```

The `etree` variable should be set to `yes` or `no`, and indicates whether the cencalvm and related libraries are available. The `CXX`, `CC`, and `FC` variables should be set to the names of the C++, C, and Fortran compilers, respectively. The `WPPROOT` variable should be set to the path of the third party libraries, such that the directories `WPPROOT/lib` and `WPPROOT/include` contain the libraries and include files, respectively. Finally, the `EXTRA_CXX_FLAGS`, `EXTRA_FORT_FLAGS`, and `EXTRA_LINK_FLAGS` variables can optionally be set to any additional argument that needs to be passed to the C++ compiler, Fortran compiler, or linker, on your system.

A.5 Installing cencalvm and its supporting libraries

The cencalvm library was developed by Brad Aagaard at USGS. Both the cencalvm library and the Etree data base files, holding the material model of Northern California, can currently be downloaded from

earthquake.usgs.gov/regional/nca/3Dgeologic/cencalvm_doc

The installation process, which is outlined below, is described in detail on the above web page. Note that three libraries need to be installed: `euclid` (`etree`), `proj4`, and `cencalvm`. In order for WPP to use them, they should all be installed in the same directory and you should assign that directory to the `WPPROOT` variable in the `make.inc` file discussed above.

Note that the `euclid` library must be installed manually by explicitly copying all include files to the include directory and all libraries to the lib directory,


```

shell> cd euclid3-1.2/libsrc
shell> make
shell> cp *.h ${WPPROOT}/include
shell> cp libetree.* ${WPPROOT}/lib

```

The proj4 library should be configured to be installed in WPPROOT. This is accomplished by

```

shell> cd proj-4.7.0
shell> configure --prefix=${WPPROOT}
shell> make
shell> make install

```

The cencalvm library should also be configured to be installed in WPPROOT. You also have to help the configure script finding the include and library files for the proj4 and etree libraries,

```

shell> cd cencalvm-0.6.5
shell> configure --prefix=${WPPROOT} CPPFLAGS="-I${WPPROOT}/include" \
               LDFlags="-L${WPPROOT}/lib"
shell> make
shell> make install

```

To verify that the libraries have been installed properly, you should go to the WPPROOT directory and list the lib subdirectory. You should see the following files (on Mac OSX machines, the .so extension is replaced by .dylib):

```

shell> cd ${WPPROOT}
shell> ls lib
libetree.so libetree.a
libproj.so libproj.a libproj.la
libcencalvm.a libcencalvm.la libcencalvm.so

```

Furthermore, if you list the include subdirectory, you should see include files such as

```

shell> cd ${WPPROOT}
shell> ls include
btree.h etree.h etree_inttypes.h
nad_list.h projects.h proj_api.h
cencalvm

```

Note that the include files for cencalvm are in a subdirectory with the same name.

Once you have successfully installed all three libraries, it is easy to re-configure *WPP* to use them. Simply edit the make configuration file (*make.inc*) according to

```

etree = yes
...

```

You then need to re-compile *WPP*. Go to the wpp main directory, remove the previous executable and object files, and re-run make:

```

shell> cd /my/installation/dir/wpp-version-2.2
shell> make clean
shell> make

```

As before, if all goes well, the “WPP lives” banner is shown after the make command is completed:

```
‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘
```

```

-----
\ \ / \ / / | _ \ | _ \
 \ \ / \ / / | | ) | | )
 \ \ / \ / / | | _ / | | _ /
 \ \ / \ / / | _ | | _ |
  \ \ / \ / / | _ | | _ |

```

```

--
| | | | \ \ / / | _ _ | / | | |
| | | | \ \ / / | | _ | | (----‘ |
| | | | \ \ / / | _ | \ \ | |
| ‘----. | | \ \ / / | _ _ .----) | |
| _ _ _ | | _ | \ \ / | _ _ _ | _ _ _ / ( _ )

```

```
‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘‘‘-.,_.,.-‘‘‘-.,_.,.=‘‘‘
```

```
*** Configuration file: /Users/andersp/src/wpp/configs/make.amac ***
```

```
*** Build directory: optimize_v2.2 ***
```

Appendix B

Testing the *WPP* installation

Once *WPP* has been installed, it is a good idea to verify that the code works properly. For this purpose, we provide test scripts in the `examples` directory. With each input file `xyz.in`, there is a corresponding output file named `xyz.out`. Note that when *WPP* runs in parallel, some of the output can appear in a different order. The most important aspect of these tests is to verify the reported errors in the numerical solutions, which is reported near the end of the output files. These numbers should be independent of the number of MPI processes on a given machine, but can vary slightly from one type of hardware to another, due to roundoff errors in floating point arithmetic. Note that some of the tests use a significant number of grid points and will only fit in memory on larger machines.

On some systems, it is necessary to start an MPI daemon before any parallel programs can be started. This is often done by issuing the command

```
mpd &
```

Refer to your local sysadmin if you have problems running `wpp` in parallel.

B.1 Method of manufactured solutions

The method of manufactured solutions (also know as twilight zone testing) provides a general way of testing the accuracy of numerical solutions of partial differential equations, including effects of heterogeneous material properties and various boundary conditions on complex geometries. The test scripts can be found in the directory

```
.../wpp-version-2.2/examples/twilight
```

In the twilight zone testing module of *WPP*, we take the material properties to be

$$\begin{aligned}\rho(x, y, z) &= A_\rho (2 + \sin(\omega_m x + \theta_m) \cos(\omega_m y + \theta_m) \sin(\omega_m z + \theta_m)), \\ \mu(x, y, z) &= A_\mu (3 + \cos(\omega_m x + \theta_m) \sin(\omega_m y + \theta_m) \sin(\omega_m z + \theta_m)), \\ \lambda(x, y, z) &= A_\lambda (2 + \sin(\omega_m x + \theta_m) \sin(\omega_m y + \theta_m) \cos(\omega_m z + \theta_m)).\end{aligned}$$

N_x	h	$\ w - w_e\ _\infty$	ratio
31	$1.667 \cdot 10^{-1}$	$1.25 \cdot 10^{-1}$	—
61	$8.333 \cdot 10^{-2}$	$3.28 \cdot 10^{-2}$	3.81
121	$4.167 \cdot 10^{-2}$	$8.67 \cdot 10^{-3}$	3.78

Table B.1: Twilight test: Max norm errors in the vertical displacement component.

The internal forcing, boundary forcing and initial conditions are chosen such that the exact (manufactured) solution becomes

$$\begin{aligned}
u_e(x, y, z, t) &= \sin(\omega(x - c_e t)) \sin(\omega y + \theta) \sin(\omega z + \theta), \\
v_e(x, y, z, t) &= \sin(\omega x + \theta) \sin(\omega(y - c_e t)) \sin(\omega z + \theta), \\
w_e(x, y, z, t) &= \sin(\omega x + \theta) \sin(\omega y + \theta) \sin(\omega(z - c_e t)).
\end{aligned}$$

The values of the material parameters (ω_m , θ_m , A_ρ , A_λ , A_μ) and the solution parameters (ω , θ , c_e), can be modified in the input script. Since the exact solution is known, it is possible to evaluate the error in the numerical solution. By repeating the same test on several grid sizes, it is possible to establish the convergence order of the numerical method.

The basic twilight tests use a single grid, a flat topography, and cover the computational domain $(x, y, z) \in [0, 5]^3$. These cases are provided in the three scripts:

```
gen-twi-1.in  gen-twi-2.in  gen-twi-3.in
```

The numerical solution is simulated up to time $t = 4.8$ on a grid with 31^3 , 61^3 , and 121^3 grid points, respectively. The corresponding results are given in the three output files

```
gen-twi-1.out  gen-twi-2.out  gen-twi-3.out
```

Assuming `mpiexec` is used to execute parallel programs, you run the first of these cases on 4 processes with the command

```
cd examples/twilight
mpiexec -n 4 ../../optimize_v2.2/wpp gen-twi-1.in
```

The errors in max and L_2 norm in the numerical solution is reported at the bottom of these files and some of these numbers are summarized in Table B.1.

To test mesh refinement on a geometry with flat topography, we provide the scripts

```
mref-twi-1.in  mref-twi-2.in  mref-twi-3.in
```

Again the numerical solution is simulated up to time $t = 4.8$ on a grid with 31^3 , 61^3 , and 121^3 grid points, respectively. A refined mesh with half the grid size is used near the free surface, in $0 \leq z \leq 2$. The corresponding results are given in the three output files

```
mref-twi-1.out  mref-twi-2.out  mref-twi-3.out
```

Non-planar free surfaces are tested by the scripts

```
gauss-twi-1.in  gauss-twi-2.in  gauss-twi-3.in
```

In this case, the free surface is a Gaussian hill and the numerical solution is simulated up to time $t = 0.8$ on a grid with 31^3 , 61^3 , and 121^3 grid points, respectively. The curvilinear grid covers the domain between the free surface and $z = 0.25$, and a single Cartesian grid covers the remainder of the computational domain ($0.25 \leq z \leq 1$). The corresponding results are given in the three output files

```
gauss-twi-1.out  gauss-twi-2.out  gauss-twi-3.out
```

Note that some image files are generated by these scripts and placed in the sub-directories `gauss_31`, `gauss_61`, and `gauss_121`, respectively. We encourage the user to look at these image files, for example by reading them into matlab/octave using the script `tools/readimagepatch.m`.

B.2 Lamb's problem

The *WPP* installation can be tested further by solving Lamb's problem, i.e., the motion due to a vertical point force applied on the free surface. The material is assumed homogeneous and the ratio between the compressional and shear velocities is $\sqrt{3}$, i.e., $\lambda = \mu$. Hence the shear velocity is $C_s = C_p/\sqrt{3}$. The time function is a `verySmoothBump` with `freq=1` and `t0=0`. Note that only the vertical component of the exact solution is available, and it is only calculated along the free surface ($z = 0$). Images of the error can be saved with the `image` command using the options `type=uzerr` `z=0`. The errors in the solution, measured in max and L_2 norms, are reported at the end of the run. This problem tests the implementation of a point force and (to some extent) the supergrid far field boundary condition. The input files can be found in the directory

```
shell> cd wpp-version-2.2/examples/lambtest
shell> ls
Lambtest1.in  Lambtest2.in  Lambtest3.in  Lambtest4.in
```

Here we provide input files with four different grid sizes, with $116^2 \times 59$, $231^2 \times 116$, $461^2 \times 231$, and $921^2 \times 461$ grid points, respectively. Be aware that the finest grid uses about 391 Million grid points and can only be run on a sufficiently large machine. The corresponding output files are given in

```
Lambtest1.out  Lambtest2.out  Lambtest3.out  Lambtest4.out
```

Note that the most important information is near the end of these files, where the error in the numerical solution is reported.

Bibliography

- [1] K. Aki and P.G. Richards. *Quantitative Seismology*. University Science Books, second edition, 2002.
- [2] D. Appelö and N. A. Petersson. A stable finite difference method for the elastic wave equation on complex geometries with free surfaces. *Comm. Comput. Phys.*, 5:84–107, 2009.
- [3] S. M. Day, J. Bielak, D. Dreger, S. Larsen, R. Graves, A. Pitarka, and K. B. Olsen. Test of 3D elastodynamic codes: Lifelines project task 1A01. Technical report, Pacific Earthquake Engineering Center, 2001.
- [4] S. M. Day, J. Bielak, D. Dreger, S. Larsen, R. Graves, A. Pitarka, and K. B. Olsen. Test of 3D elastodynamic codes: Lifelines program task 1A02. Technical report, Pacific Earthquake Engineering Center, 2003.
- [5] H. Emmerich and M. Korn. Incorporation of attenuation into time-dependent computations of seismic wave fields. *Geophysics*, 52(9):1252–1264, 1987.
- [6] P. Goldstein, D. Dodge, M. Firpo, and L. Miner. *International Handbook of Earthquake and Engineering Seismology*, volume 81B, chapter SAC2000: Signal processing and analysis tools for seismologists and engineers, pages 1613–1614. International Association of Seismology and Physics of the Earth’s Interior, 2003.
- [7] B. Gustafsson, H.-O. Kreiss, and J. Oliger. *Time dependent problems and difference methods*. Wiley–Interscience, 1995.
- [8] H.-O. Kreiss and N.A. Petersson. Boundary estimates for the elastic wave equation in almost incompressible materials. *SIAM J. Numer. Anal.*, 50:1556–1580, 2012.
- [9] H. Lamb. On the propagation of tremors over the surface of an elastic solid. *Phil. Trans. Roy. Soc. London, Ser. A*, 1904.
- [10] P. Liu, R. J. Archuleta, and S. H. Hartzell. Prediction of broadband ground-motion time histories: Hybrid low/high-frequency method with correlated random source parameters. *Bulletin of the Seismological Society of America*, 96:2118–2130, 2006.
- [11] H. M. Mooney. Some numerical solutions for Lamb’s problem. *Bulletin of the Seismological Society of America*, 64, 1974.
- [12] S. Nilsson, N.A. Petersson, B. Sjögreen, and H.-O. Kreiss. Stable difference approximations for the elastic wave equation in second order formulation. *SIAM J. Numer. Anal.*, 45:1902–1936, 2007.

- [13] N. A. Petersson and B. Sjögreen. Stable grid refinement and singular source discretization for seismic wave simulations. *Comm. Comput. Phys.*, 8(5):1074–1110, November 2010.
- [14] N. A. Petersson and B. Sjögreen. Stable and efficient modeling of anelastic attenuation in seismic wave propagation. *Comm. Comput. Phys.*, 12(1):193–225, 2012.
- [15] P. Wessel and W. H. F. Smith. New, improved version of generic mapping tools released. In *EOS trans. AGU*, volume 79, page 579, 1998.

Index

- attenuation, 38
- block parameters
 - vp, vs, rho, qp, qs, vpgrad, vsgrad, rhograd, x1, x2, y1, y2, z1, z2, absdepth, 71
- boundary_conditions parameters
 - lx, hx, ly, hy, lz, hz, 85
- boundary_conditions values
 - clayton-engquist, energy-absorbing, stress-free, dirichlet, neumann, supergrid, 85
- command
 - attenuation, 69
 - block, 70
 - boundary_conditions, 84
 - developer, 85
 - efile, 71
 - fileio, 64
 - globalmaterial, 74
 - gmt, 82
 - grid, 65
 - ifile, 73
 - image, 77
 - material, 73
 - pfile, 72
 - prefilter, 68
 - refinement, 75
 - sac, 76
 - source, 66
 - supergrid, 84
 - testlamb, 82
 - testpointsource, 83
 - time, 66
 - topography, 74
 - twilight, 82
 - volimage, 79
- command line options
 - v version info, 8
- coordinate system, 9
- developer parameters
 - cfl_number, interpolation, ctol, cmaxit, 86
 - output_load, output_timing, log_energy, print_energy, mpiio, iotiming, 86
- efile parameters
 - logfile, vsmin, vpmmin, query, resolution, access, etree, xetree, 72
- examples, 47
 - caucasus, 59
 - earthquake, 56
 - grenoble, 53
 - lambs, 47
 - scec, 48
- fileformats, 87
 - ifile, 91
 - image, 93
 - pfile, 89
 - sac, 92
 - topography, 87
 - volimage, 94
- fileio parameters
 - path, verbose, printcycle, pfs, nwriters, 65
- geographical coordinates, 10
- globalmaterial parameters
 - vpmmin, vsmin, 74
- gmt parameters
 - file, 82
- grid parameters
 - location - az, lat, lon, 66
 - size - x, y, z, h, nx, ny, nz, 66
- grid size, 21
- gridsize, 12
- ifile parameters
 - filename, 73
- image mode options

- lat, lon, topo, grid, 78
- rho, lambda, mu, p, s, qp, qs, 78
- ux, uy, uz, div, curl, veldiv, velcurl, velmag,
 - hvel, hvelmax, vvelmax, uxerr, uyerr, uzerr, fx, fy, fz, 79
- image parameters
 - file, mode, precision, 78
 - location - x, y, z, 77
 - timing - time, timeInterval, cycle, cycleInterval, 78
- installation, 99
 - basic, 102
 - cencalvm, 103
 - directories, 101
 - efile, 103
 - platforms, 99
 - tools, 99
- material, 24
- material parameters
 - id, vp, vs, rho, qp, qs, 73
 - vp2, vs2, rho2, vpsqrt, vssqrt, rhosqrt, 74
 - vpgrad, vsgrad, rhograd, 73
- mesh refinement, 35
- output options, 41
- parallel execution, 7
- performance, 96
- pfile parameters
 - filename, directory, smoothingsize, vpmmin, vsmin, rhomin, flatten, style, 72
- prefilter parameters
 - fc, maxfreq, 69
- refinement parameters
 - zmax, 76
- sac parameters
 - location - x, y, z, lat, lon, depth, topodepth, 77
 - sta, file, type, writeEvery, eventDate, eventTime, nsew, velocity, usgsformat, sacformat, 77
- source parameters
 - Aki and Richards - strike, dip, rake, 68
 - location - x, y, z, depth, topodepth, lat, lon, 67
 - moment - m0, mxx, myy, mzz, mxy, mxz, myz, 68
 - point force - f0, fx, fy, fz, 68
 - t0, freq, type, 67
- source time dependence
 - GaussianInt, Gaussian, RickerInt, Ricker, Ramp, Triangle, Sawtooth, Smoothwave, VerySmoothBump, Brune, BruneSmoothed, GaussianWindow, Liu, 67
- sources, 12
- srun, 7
- supergrid parameters
 - thickness, damping, 84
- testing, 106
 - lambs, 108
 - twilight, 106
- testlamb parameters
 - x, y, cp, rho, fz, 83
- testpointsource parameters
 - m0, mxx, mxy, mxz, myy, myz, mzz, f0, fx, fy, fz, 84
 - x, y, z, cp, cs, rho, freq, t0, type, 83
- time parameters
 - t, steps, 66
- topography, 31
- topography parameters
 - gaussianAmp, gaussianXc, gaussian Yc, gaussianLx, gaussianLy, 75
 - input, file, resolution, zmax, order, smooth, 75
- twilight parameters
 - errorlog, omega, c, phase, momega, mphase, amprho, ampmu, amplambda, 82
- units, 9
- volimage options
 - cycle, cycleInterval, time, timeInterval, startTime, x1, x2, y1, y2, z1, z2, 81
 - file, mode, sample, precision, 80